TRANSVERSION SYSTEM
AN APPLE II TO COMMODORE 64 FILE TRANSFER/CONVERSION SYSTEM

By

Lonald L. Fink
B.S.E.E., Purdue University, 1976

A Thesis
Submitted to the Faculty of the
University of Louisville
Speed Scientific School
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Electrical Engineering

August 1987

TRANSVERSION SYSTEM
AN APPLE II TO COMMODORE 64 FILE
TRANSFER/CONVERSION SYSTEM

Submitted by: ______________________________
Lonald L. Fink

A Thesis Approved on

July 23, 1987
____
Date

by the Following Reading and Examination Committee:

____
Thesis Director, Dr. Thomas G. Cleaver

____
Dr. William H. Pierce

____
Dr. Waldemar Karwowski

ACKNOWLEDGEMENTS

The author thanks Dr. Thomas G. Cleaver for his leadership and supervision of this project, and the author expresses his appreciation for the love and support from his fiancee, Carol M. Thomas.

He also appreciates the contributions of a long time friend Thomas Robertson, for his help in the understanding of the operations of the various kernal routines used by Commodore Basic. Finally, he wishes to thank Kenneth W. Harper II, his supervisor at NOSL (Naval Ordnance Station Lousiville), for his understanding and approval of the author's liberal use of annual leave which was necessary for the latter's finishing of his project.

ABSTRACT

This thesis describes the design, development, and testing of the TransVersion system, a hardware and software solution in the development of Commodore 64 software by TRANSfer and conVERSION of Apple II software. The TransVersion system is able to transfer Apple DOS 3.3 files which include binary files, Applesoft Basic program files, sequential text files, and random access files. The software consists of the transfer and conversion software necessary to transfer Apple II files to the Commodore 64, and an Apple II Basic emulation software package for the Commodore 64 to assist in the running of the transferred Apple II software on the Commodore 64. The hardware consists of a simple cable connecting the Apple II paddle port to the Commodore 64 User port. The TransVersion system was found to quickly and easily transfer and convert Apple II disk files to the Commodore 64 computer. A 300 line program was transferred in about 12 seconds, but required additional conversion time of 48 seconds.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. Background

The Commodore 64 (C-64) computer is relatively new to the personal computer market. As with most new personal computers, the quality software needed to use the computer usually lags behind the introduction of the host computer. The C-64 is no exception. The quantity of software available to the C-64 is small compared to the software available for the Apple II series computers, which were introduced several years before by Apple Computer, Inc.

Computer software lags behind the introduction of the host computer because of the lengthy amounts of time needed by programmers to generate the programs needed on the host computer. Some computer programmers will try to circumvent this long process of generating the new software by converting the software available on other computers to the host computer. In the case of the C-64 computer, the Apple II computer series is a logical choice to borrow existing software for conversion to the C-64. The Apple II computer and the C-64 computer are similar in many hardware respects. Both computers have compatible 8 bit central processing units (CPU), a 6502 and a 6510 (enhanced 6502), nearly identical main clock speeds, and

similar ROM based Basic operating systems, Applesoft and Commodore Basic. The Apple II series computers also have thousands of programs available free in the public domain to the user for his/her selection. Table I summarizes some of the differences between the Commodore 64 and the Apple II computers.

TABLE I

APPLE II AND COMMODORE 64

HARDWARE AND SOFTWARE COMPARISONS

DESCRIPTION	APPLE II	COMMODORE 64
Quantity of Software	large	small
CPU Type	6502	6510 (6502 compatible)
Memory Size	48K/64K	64K
Clock Speeds	1,023,000 Hz.	1,000,000 Hz.
Resident Language	Applesoft	Commodore Basic

B. Definition of Problem

Although the manual conversion of programs from the Apple II computer to the C-64 computer is a viable solution, problems with converting programs from one computer to another computer can occur. Table II lists the major problems.

TABLE II

LIST OF CONVERSION PROBLEMS

1. Different command words are used for the same function.

2. Same command words have different functions.

3. Command words on source computer may have no equivalent on target computer.

4. Different syntax rules govern common command words.

5. Conversion of programs "by hand" is time consuming and error prone.

6. Different floppy disk formats between the Apple II and the Commodore 64

With the different diskette software formats being used with the Apple II and the C-64 diskettes, the programmer must resort to some other means of transferring the Apple software programs to the C-64 computer besides a simple Basic LOAD command. The most common used method is to type in the borrowed Apple software into the host computer (C-64). The most common problem in this method of converting programs is the human errors created when typing in the borrowed programs into the memory of the host computer. Occasionally programmers will use a variety of expensive technological devices and software packages to transfer data to the host computer. These devices might include modems, RS232 ports, or parallel

interfaces. These devices eliminate the common typing errors associated with entering programs into the host computer. However this method is costly and enters the data into the host computer in exactly the same format as the source data with no conversion capabilities.

The problems shown above encountered by the programmer when converting programs from one computer to the host computer can be solved. This thesis documents the TransVersion System - An Economical Apple II to Commodore 64 File Transfer/Conversion System which is an efficient solution to the above discussed problems.

C. Overview of the TransVersion System and its Advantages

1. Design Criteria

The TransVersion System will provide to the programmer an efficient economical process to convert Apple II Basic software to C-64 software. The major reason that the TransVersion System works more efficiently is the fact that a semiautomatic operation can be more efficient and usually less error prone than a manual operation. Based on this fact, the TransVersion System was created using the criteria listed in Table III.

2. TransVersion System Description

The TransVersion System consists of 4 major

components as shown in Table IV.

a. <u>Hardware Interface Description</u> Design criterion 1, listed in Table III requires some sort of data transfer from the Apple II computer to the C-64 computer. Although there are a variety of computer communication devices available, it was decided (for purely economic reasons) to connect directly the Apple II computer to the Commodore 64. The Apple II paddle port and the C-64 User port are ideal for this purpose.

TABLE III

MAJOR DESIGN CRITERIA FOR THE TRANSVERSION SYSTEM

1. Use a simple hardware interface to keep economic cost to a minimum and which will allow data transfer from an Apple II computer to a C-64 computer to occur quickly, thus eliminating any human typing errors.

2. Use a semiautomatic conversion process to eliminate as much as possible the manual editing of the converted programs on the C-64 computer.

3. Implement as many as possible of the unavailable Applesoft commands on the C-64 computer using emulation program techniques which will allow the maximum amount of the converted Apple II programs to run on the C-64.

4. Identify and comment out (REM) all incompatible commands for easy identification and correction when editing the transferred programs.

5. Make available software options to allow the associated Apple II data files of the converted Apple II programs to be transferred from the Apple II to the C-64.

TABLE IV

MAJOR COMPONENTS OF THE TRANSVERSION SYSTEM

1. A simple cable interface to allow transfer of data from the Apple II computer to the C-64 computer.

2. Data transfer driving software for the Apple II computer so that data may be sent to the C-64 computer from the Apple II computer.

3. Data transfer/conversion receiving software for the C-64 so that data may be received from the Apple II computer and converted into C-64 format.

4. Apple II Basic command emulation software for the C-64 so that unavailable Applesoft commands can be implemented.

These two ports are buffered, and the TTL logic voltages are compatible.

By using a three wire cable with compatible connectors, a simple serial data transfer system with handshaking was implemented. A full description of the system hardware is detailed in Chapter II.

b. Apple II driver software Using an Applesoft Basic menu program and a variety of machine language transfer driver programs, an Apple II transfer driver software package was implemented to allow all types of Apple II files to be sent to the C-64 computer via the Apple II paddle port interface. Chapter III gives a full description of this software. A complete documented source listing may be found in Appendix B.

c. Commodore 64 Receiving Software The receiving software is loaded into the C-64 memory using a menu driven C-64 Basic program. This software allows the C-64 to receive data from the Apple II via the C-64 User port interface. The receiving software will convert the received data into the proper C-64 syntax, comment out (REM) incompatible commands, store the resultant program to the diskette, and return to the menu program. A full description of this software can be found in chapter III. A complete documented source listing may be found in Appendix C.

d. Apple II Commands Emulation Software This software was created to fulfill design criterion 3 listed in Table III. This software is loaded when attempting to run or edit converted programs with embedded emulated commands. This software allows the C-64 to run converted Applesoft programs that would not normally run using Commodore Basic. A full description of this software is given in Chapter III. A complete documented source listing may be found in Appendix D.

II. TRANSVERSION SYSTEM HARDWARE DESCRIPTION

A. Apple II Paddle Port

The Apple II paddle port has two sections of interest: the annunciator outputs and the pushbutton inputs. The Apple paddle port is connected via a 16 pin dual-in-line socket. The pin-out of the Apple paddle port is shown in Figure 1.

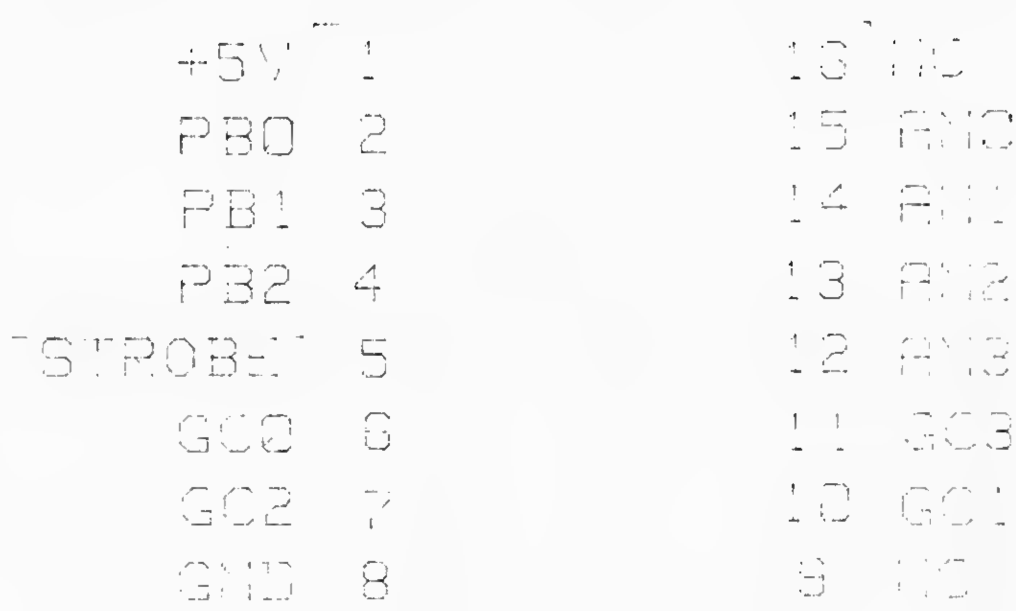

FIGURE 1 - Apple II Paddle Port Pin-Out

1. Annunciator Outputs

The annunciator outputs consist of 4 output pins that may be toggled to a logic high or low voltage by writing to consecutive memory addresses. The annunciator outputs can be changed by the appropriate software command. In Basic the POKE command is used. In assembly language the store command (e.g. STA,STX,STY) is used. The value stored in memory is not important, but where the value is stored determines the output state of the annunciator. The annunciator output addresses and their associated output states are listed in Table V.[2]

TABLE V

ANNUNCIATOR MEMORY I/O ADDRESSES

Paddle Port Socket Pin	Annunciator Name	Logic State	Memory Address Hex	Memory Address Decimal
15	AN0	0	C058	49240
15	AN0	1	C059	49241
14	AN1	0	C05A	49242
14	AN1	1	C05B	49243
13	AN2	0	C05C	49244
13	AN2	1	C05D	49245
12	AN3	0	C05E	49246
12	AN3	1	C05F	49247

The state of the annunciator output cannot be

determined by the program, therefore the annunciator output must be initially set by the programming software. See Apple II software description in Chapter III for more details on the programming of the annunciator outputs. The annunciator output is a transistor-transistor logic (TTL) digitally produced voltage output (74LS259) and is not designed for high current operation.[3] Since the annunciator outputs are connected to the C-64's User port TTL inputs via the TransVersion Interface cable, no hardware buffer chips are needed. However, TTL logic circuits are not well buffered against incompatible grounds, which is discussed later in this chapter. The TransVersion system uses only annunciator output AN1. Annunciator output AN1 is used as a "data out" line to send data as a stream of bits to the C-64.

2. <u>Pushbutton Inputs</u>

The pushbutton inputs of the Apple II paddle port are used to allow data to be entered into the Apple II. The pushbutton inputs are connected internally to the most significant bit 7 (MSB) of the corresponding pushbutton input I/O memory address via a 74LS251 chip. Table VI lists the memory addresses and their corresponding Pushbutton inputs.[4]

TABLE VI

PUSHBUTTON INPUT I/O MEMORY ADDRESSES

Paddle Port Socket Pin	Pushbutton Name	Memory Address Hex	Memory Address Decimal
2	PB0	C061	49249
3	PB1	C062	49250
4	PB2	C063	49251

Since the pushbutton input is a single bit input, all lower order bits (bit 0 - bit 6) are meaningless and should be ignored when reading the pushbutton memory addresses. The logic state of the MSB is determined by the voltage on the pushbutton input pin. A voltage of five volts on pushbutton input pin PB0 gives a high logic state (logic 1) at the MSB of memory address 49249. A zero voltage on pushbutton input pin PB0 gives a low logic state (logic 0) on the MSB of memory address 49249. Since the MSB of a memory address sets the N flag in the status register of the CPU when a read operation occurs, in assembly language a simple Branch on Minus (BMI) statement can determine the state of the pushbutton input pin after a read operation of the appropriate memory address. If the branch is taken the pushbutton input is in a high state (5 volts). If the branch is not taken a low state (0 volts) is present on the pushbutton input pin. In

Basic, the PEEKed value can be compared with 128; if the PEEKed value is lower than 128 then a low voltage is present on the pushbutton input pin, and if the PEEKed value is greater than or equal to 128, then a high voltage is present on the pushbutton input pin.[5] The pushbutton input PB1 is used by the TransVersion system to determine if the C-64 computer is ready to accept input data.

B. Commodore 64 User Port

The Commodore User port is connected to the 6510 CPU via a 6526 Complex Interface Adapter (CIA) chip. This TTL compatible chip is used to interface a variety of devices to the 6510 CPU inside the C-64. The TransVersion system use of the C-64 User port will conflict with many standard uses of the C-64 User Port. See Appendix A - The TransVersion User Guide for more details on what conflicts that might occur when using the C-64 User port with the TransVersion System interface cable.

The 6526 CIA chip can be programmed to allow the 8 I/O lines to be either inputs or outputs. The CIA chip can be interrupt driven or polled for information. The pin-out of the C-64 User port is given in Figure 2.[6] The connector needed to attach to this port is a 24 pin edge card connector with 0.156 inch pin spacing and opposite pin spacing that will accept a 0.0625 inch thick printed circuit board.

GND	A	1	GND
FLAG2	B	2	+5V
PB0	C	3	RESET
PB1	D	4	CNT1
PB2	E	5	SP1
PB3	F	6	CNT2
PB4	H	7	SP2
PB5	J	8	PC2
PB6	K	9	ATN
PB7	L	10	9VAC+
PA2	M	11	9VAC-
GND	N	12	GND

FIGURE 2 - C-64 User Port Pin out

The Data Direction Register of port B (DDRB) of the CIA chip addressed in the C-64 computer at I/O memory address 56579 ($DD03 hex) controls the direction of the eight I/O lines (PB0 thru PB7) in port B at I/O memory address 56577 ($DD01 hex). Each of the eight lines in port B has a bit in the eight bit Data Direction Register (DDRB) which controls whether that line will be an input or output. By setting the corresponding bit in the DDRB the programmer can set any line in port B to an output line. By clearing the corresponding bit in the DDRB, the programmer can set any line in port B to an input line. After the DDRB is set, port B address ($DD01 hex) may be written to send data out port B or a read from port B address ($DD01 hex) to receive data in from port B.[7]

The TransVersion system uses two of the eight possible I/O lines in port B (PB0 and PB1). PB0 is set to an input line. PB1 is set to an output line. The PB0 input line is used to receive the data from the Apple II. The PB1 output line is used to inform the Apple II that the C-64 is ready to receive the data to be transmitted.

The PB0 line of port B corresponds to the least significant bit (LSB) of I/O memory address $DD01 hex. By examining the LSB after a read operation of the I/O address of port B, the C-64 can determine the data being

received from the Apple II. See Chapter III for more detailed programming information on port B of the C-64 User port.

C. <u>TransVersion Interface Cable</u>

The TransVersion Interface cable is a three wire cable which is used to connect the Apple II paddle port to the C-64 User port. The Apple paddle port and the C-64 User port hardware are not designed with line driver/ receiver hardware; thus the length of the interface cable must be kept short. A maximum of ten feet in length is allowed for the interface cable. Due to TTL's low voltage levels (max. 5 volts) and possible grounding problems as discussed in detail later in this chapter, cable lengths longer than ten feet could cause serious signal degradation and possible transfer errors from noise.

The TransVersion Interface cable has compatible hardware connectors for the Apple II paddle port and the C-64 User port at opposite ends. The Apple II paddle port compatible plug is a 16 pin dual-in-line header plug which will connect directly to the Apple II paddle port dual-in-line socket. The C-64 User port compatible plug is a 24 pin edge card connector with a 0.156 inch pin spacing which will connect directly to the C-64 User port.

The TransVersion interface cable has the function

of connecting the appropriate signals at the Apple II paddle port and the C-64 User port. Table VII details the wiring diagram of the TransVersion Interface cable.

TABLE VII

TRANSVERSION INTERFACE CABLE WIRING LIST

Apple II Paddle Port Connection			C-64 User Port Connection		
Pin Number	Type	Pin Name	Pin Number	Type	Pin Name
8	--	GND	A	--	GND
14	OUT	AN1	C	IN	PB0
3	IN	PB1	D	OUT	PB1

The ground connections (pins 8 and A) are included to give the Apple II computer and the C-64 computer a common ground at which to reference the signals shown in Table VII. Although the ground connections will eliminate some ground problems (e.g. open ground), the ground connection (20 gauge diameter wire or smaller) will not eliminate high voltage potential differences between the grounds of the computers. Therefore, it is suggested to use the same 3 prong wall socket or a power strip to power both computers to reduce or eliminate any ground voltage differences that might occur on different wall outlets due

to the resistance of the wires and high current flows in the 115 V AC power system's ground wires inside the residence or business.

The AN1 (pin 14) and PB0 (pin C) interface connection is used to carry the signals which transfer the data to the C-64. The PB1 (pin D) and PB1 (pin 3) interface connection is used to carry the signals to start the transfer of data simultaneously on both computers; this is a handshaking line. This simple interface allows the transfer of data from the Apple II computer to the C-64 computer.

III. TRANSVERSION SYSTEM SOFTWARE DESCRIPTION

A. Apple II Computer Program Descriptions

The Apple driver routines are designed to send the specified Apple file data to the C-64. The actual data transfer routines are written in assembly language for speed and efficiency. The Apple 'HELLO' program is an Applesoft Basic program that will allow the transfer procedure to be user friendly. The HELLO program will direct the user throughout the transfer process. It will LOAD the necessary software, display error messages, prompt the user for necessary information and inform the user when the transfer is complete. The HELLO program will request from the user the information listed in Table VIII.

TABLE VIII

USER OPERATIONS REQUESTED BY THE HELLO PROGRAM

1. Entry of the name of the file.
2. Entry of the type of file.
3. Entry of the record length if the file type selected is random access.
4. Insertion of the proper diskettes at the proper times.

From the information received from the user the Hello program will perform the operations listed in Table IX.

TABLE IX

HELLO PROGRAM OPERATION LIST

1. Perform a Basic LOAD command to load the necessary machine language subroutine software into memory.

2. Store into memory the necessary information needed by the machine language programs.

3. Create an EXEC file named TRANSFER BASIC to control the transfer process during an Applesoft Basic file transfer.

4. Perform a Basic CALL command to execute the machine language program to do the transfer.

5. After the transfer, generate and display appropriate messages which are used to inform the user of the transfer status.

6. Return control of the Apple to the user.

Although the Apple HELLO program allows for user friendly transfer procedures, the way the transfer is accomplished is determined by the various machine language programs loaded into memory by the HELLO program. The loading of which machine language program is determined by

the type of file that is specified to be transferred. The machine language program that is loaded into the Apple computer memory determines the different transfer procedures. The transfer procedures are different because of the different file structures of the different type of files available in Apple DOS 3.3 and because of the order of the development of the different software transfer routines. The different file structures will cause minimum software differences in an all encompassing file type transfer software program (i.e. a generic file type transfer routine). However, the order in which the transfer programs were developed caused distinct differences in the transfer software programs for each type of file. In the initial development of the Applesoft Basic file transfer program, the simplest procedure was developed for the transfer program. This transfer procedure was developed for Applesoft Basic programs with no considerations taken into account for the other types of files (e.g. text, binary). These file types was not considered for the development of software transfer programs until the completion of the Applesoft Basic transfer routine. The transfer of type files other than Applesoft Basic files was not considered necessary until the author's realization that some Applesoft Basic programs (e.g. word processing programs) were useless

without the accompanying Applesoft Basic data files (e.g. text files, binary files). Thus, the binary file transfer software routine and text file transfer software routines were developed almost simultaneously and are very similar in the way they operate. Although the Applesoft Basic transfer routine and the other file type transfer routines are distinctly different, a generic file type transfer routine could be developed that would transfer all different file types with a procedure similar to the random access text file transfer routine procedure. The all encompassing transfer routine was not developed for each file type because of the additional transfer software development time that would be necessary.

The descriptions of the transfer process and the machine language programs that do the transfer are separated into four different sections determined by the type of file to be transferred which are described below.

1. Applesoft Basic File Transfer

Before an Applesoft Basic file transfer proceeds, the Apple HELLO program will LOAD the machine language program MLBASICTRANSFER into memory, create an EXEC file named TRANSFER BASIC on the diskette containing the file to be transferred, and SAVE itself as the file named MASTER TRANSFER.

The EXEC file 'TRANSFER BASIC' will LOAD the specified Applesoft Basic file to be transferred and CALL the machine language program MLBASICTRANSFER residing in memory.

The machine language program MLBASICTRANSFER, which is stored in the cassette buffer starting at memory location 768 ($300 hex),[8] will transfer the program data to the C-64 a byte at a time. The MLBASICTRANSFER program will first send the name of the program and the file type to the C-64. The MLBASICTRANSFER program will retrieve and send to the C-64 data starting from the memory location pointed to by the start of Basic pointer at locations 103 and 104 ($67 and $68 hex).[9] The MLBASICTRANSFER program will retrieve and send the data from successive memory locations until three consecutive zeros are encountered. These three zeros mark the end of the Basic program.

The transfer of the byte data to the C-64 is done by a subroutine within the MLBASICTRANSFER routine named SEND. The SEND routine will perform the operations listed in Table X.

The SEND routine disables interrupts (SEI) to inhibit unwanted delays caused by interrupt processing of keyboard and video interrupts. The SEND routines waits for a high logic voltage to be received from the C-64.

TABLE X

OPERATIONS PERFORMED BY THE SEND ROUTINE

1. Disable all interrupts (SEI) in the Apple, including video and keyboard interrupts except simultaneous control - reset key presses.

2. Wait for the transmission enable signal from the C-64.

3. Start data transmission by sending a start bit - a logical low value.

4. Set the proper data rate for transmission.

5. Send the logical value of each bit of the data to the C-64, least significant bit first, by writing to the appropriate paddle port memory locations ($C05A and $C05B hex).

6. End the data transmission by sending a stop bit - a logical high value.

7. Enable interrupts (CLI).

This is done by monitoring (polling) the pushbutton memory location PB1 ($C062 hex). When the MSB of the pushbutton memory location goes to a one state (high), it means that a high logic voltage has been received from the C-64. As soon as the high logic voltage is received, the start of transmission is synchronized by sending a low logic level (start bit) to the C-64. A sending of a bit is a matter of holding the specified high or low logic voltage level on the interface for a specified period of

time. This is accomplished by writing to the appropriate paddle port memory location and delaying a specified period of time before writing to the next memory location. Writing to the memory location $C05B hex will produce a high voltage level. Writing to the memory location $C05A will produce a low logic voltage.[10] A full description of the hardware is in Chapter II.

After a specified time delay from sending the start bit, the least significant bit value is sent. Then the next least significant bit is sent. This process continues until all eight bits of the data byte have been sent. After the eight data bits have been sent, two stop bits (at high logic level) are sent. The high logic voltage levels of the stop bits end transmission synchronization until another start bit (low logic voltage) is sent. Interrupts are then enabled with a CLI instruction.

The above describes the transmission of data to the C-64 which involves the generation of a train of bit pulses corresponding to the data being sent. The transmission process is an asynchronous process with one start bit to synchronize the start of one character of data being transmitted and two stop bits to end its synchronization. Figure 3 shows a typical pulse train generated for the single character 'A'.

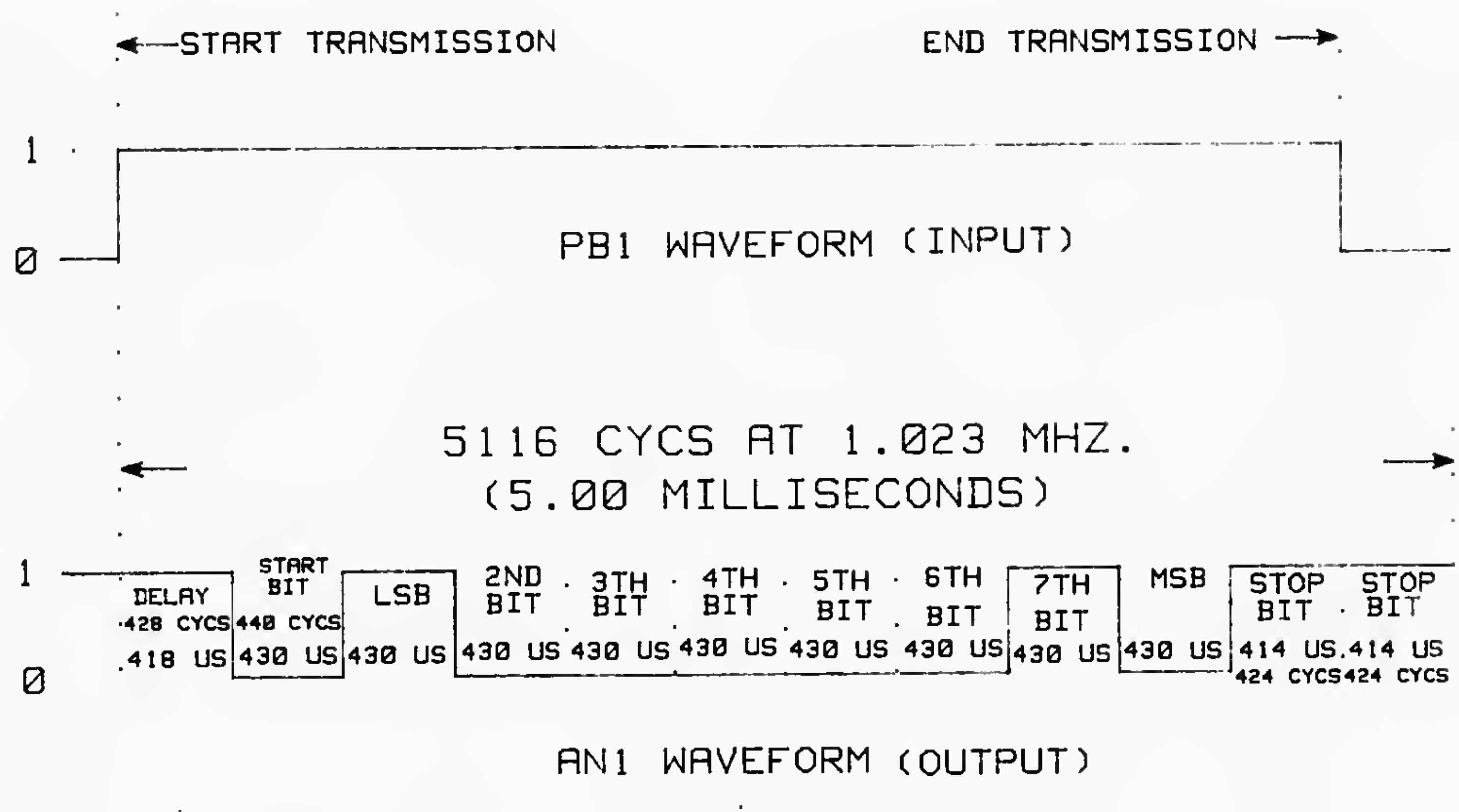

FIGURE 3 - A TYPICAL DATA STREAM FOR THE LETTER A (DATA VALUE OF 65)

The width of the pulses generated by a software delaying routine determines the maximum data transfer rate that is possible and enables the synchronization of data bits to be possible. As shown in Figure 3, the time needed to send one character is five milliseconds. By taking the inverse of this value the maximum data rate can be calculated. This results in a maximum transmission rate of 200 characters per second or a baud rate (bits/seconds) of 2200 baud. However, the effective transmission rate is much slower because of the overhead time needed to retrieve the correct data to be sent.

The Apple data values are sent directly to the C-64 with no data conversions. All conversions of data, if needed, are performed in the C-64; this is done in the C-64 rather than the Apple because of the C-64's superior editor used in the development of the software to perform the conversion, and because of the additional RAM memory available in the C-64 for storing the conversion software. The conversion process is described in the C-64 transfer section of this chapter.

After the data has been transferred, the EXEC file will LOAD and RUN the MASTER TRANSFER program that was saved on the diskette, and another file may then be transferred.

2. Binary File Transfer

Before a binary file transfer can proceed the Apple HELLO program must LOAD the machine language binary file transfer program MLBINTRANSFER into memory, LOAD the specified binary file to be transferred into memory at location 10000, and CALL the machine language program MLBINTRANSFER.

The MLBINTRANSFER program retrieves the starting address and the length from the diskette. These values are sent to the C-64 using the SEND routine described in the previous section. The new end location is calculated by adding the length value to the new start location at memory location 10000. The MLBINTRANSFER program then sends to the C-64 the specified binary file data which is stored from the new start memory location to the calculated end memory location one byte at a time using the SEND routine. After all binary data has been transferred, the Apple HELLO program regains control and another file may be transferred.

3. Sequential Text File Transfer

In order to transfer a sequential text file, the Apple HELLO program will LOAD the machine language program MLTEXTTRANSFER into memory, set the type flag to sequential, set the record length flag to zero

(i.e. non-existant) and CALL the machine language program MLTEXTTRANSFER.

The MLTEXTTRANSFER program will send the name and the type of file to the C-64. The MLTEXTTRANSFER program will retrieve the first data sector from the diskette. The first sector and subsequent sectors will then be transferred to the C-64 a byte at a time using the SEND routine described previously. Every character byte will be displayed on the screen a byte at a time during the transfer. The end of transfer will be signaled by the end of file pointer which is a zero in the data being sent to the C-64.[11] At this time, the MLTEXTTRANSFER program returns control to the HELLO program and another file may be transferred.

By retrieving the text file data a sector at a time, long sequential files may be transferred with minimum memory requirements needed in the Apple computer.

4. <u>Random Access File Transfer</u>

In order to transfer a random access text file, the Apple HELLO program will LOAD the machine language program MLTEXTTRANSFER into memory, set the type flag to random access, set the record length flag to the record length and CALL the machine language program MLTEXTTRANSFER.

The MLTEXTTRANSFER program will retrieve the first

data sector and subsequent sectors one sector at a time from the diskette. The record number and record position of the first valid data byte (non-zero byte) is calculated and sent to the C-64. The present record and subsequent records are sent to the C-64 and displayed on the screen, a byte at a time, until a zero occurs in the data. The occurrence of a zero signals that a new record number and record position will be sent to the C-64 following the zero data value. A zero data byte in a text file indicates an empty record or empty record positions. The new record number and record position are calculated for the next non-zero data value. The new record position and record number values will be sent to the C-64. The new record data is then sent. This process is repeated until the end of file is reached.

The end of file occurs when there are no more sectors available for retrieval from the diskette. This is determined by a track/sector pair pointer of zero in the diskette directory.[12] At the end of the file four consecutive zeros are sent to the C-64 to signal the end of the file. The MLTEXTTRANSFER program returns control to the HELLO program and another file transfer may occur.

B. Commodore 64 Transfer Program Description

The Commodore 64 receive routines are designed to

receive the specified Apple II file data from the Apple II. The actual data transfer routines are written in assembly language for speed and efficiency. The C-64 'MENU' program is a Commodore 64 Basic program that will allow the loading of the various software programs to be a user friendly procedure. The C-64 MENU program is very similar to the Apple Hello program in the way the program operates. The MENU program will direct the user throughout the software loading process. It will load the necessary software, automatically disable emulation mode, prompt the user for necessary information, and inform the user when the software installation is complete. The MENU program will request from the user the four items listed in Table XI.

From the information received from the user, the MENU program will perform the four operations listed in Table XII. The TransVersion User Guide (See Appendix A) can help the user with the desired responses to the requests made by the MENU program. Furthermore, on screen information assists the user in making theses choices.

Although the C-64 MENU program allows for a user friendly software installation procedure, the way the transfer is accomplished is determined by the machine language program loaded into memory by the MENU program.

TABLE XI

USER OPERATIONS REQUESTED BY THE MENU PROGRAM

1. What mode of operation does the user which to enter, transfer mode or emulation mode?

2. If transfer mode is selected, is it a Basic file that is to be transferred?

3. If the file to be transferred is a Basic file, does the user wish the Apple character set lines or Emulation lines included in the transferred program?

4. If the emulation mode is selected, does the user wish the Apple character set option installed?

TABLE XII

MENU PROGRAM OPERATIONS LIST

1. Load the necessary machine language subroutine software into memory.

2. Store into memory the necessary information needed by the machine language program.

3. Perform the Basic command SYS to execute the machine language program to do the transfer of a file from the Apple II to the C-64 or install the emulation software in the C-64 Basic operating system.

4. After the transfer, display error messages if any, or display the diskette directory after the emulation mode software installation.

After loading the machine language transfer program the Menu program gives control to the transfer program. The

transfer program will allow the C-64 to receive the information from the Apple.

The C-64 transfer program will receive the name, type, record length if needed, and the data content of the files sent from the Apple. The transfer procedures differ depending on the type of file to be transferred.

The C-64 transfer program will first retrieve the name of the file to be transferred from the Apple. The file type will then be received from the Apple. At this point the file type will determine the transfer procedure. The four different transfer procedures will be described separately by file type.

1. Binary File Transfer Procedure

The binary file transfer section of the C-64 transfer program will first receive the start address and length (i.e. number of bytes in the file) of the binary file to be transferred from the Apple. From this information, the C-64 transfer program determines if the Binary program can be stored in the same C-64 memory address locations as in the original memory address locations of the Apple. If the binary file cannot be stored in its original memory locations, the C-64 transfer program will relocate and store the binary file in the Basic program space starting at memory location 2049 ($801 hex).[13] The relocation of the program could possibly

affect the function of the program being transferred, however the data is intact and will be accurately relocated. The TransVersion system will generate and display a message to inform the user the binary data is being relocated. The C-64 transfer program will retrieve the binary file data from the Apple a byte at a time until all binary data is retrieved from the Apple and stored into C-64 memory. Transfer terminates when the number of binary data bytes received from the Apple equals the length of the binary file. After the C-64 transfer program retrieves the binary data from the Apple II, the binary file data will be stored on the diskette under the original file name with a start address from which it was stored in the C-64, the original Apple II start address or at the new relocated memory address. The C-64 transfer program will LOAD and RUN the MENU program from the diskette and another file may be transferred.

The individual data bytes are retrieved from the Apple using the CHAR routine. The CHAR routine operations are shown in chronological order in Table XIII.

The operations listed in Table XIII ensure proper synchronization of data flow from the Apple to the C-64. The CHAR routine first disables interrupts (SEI) to eliminate any delays that might be caused by interrupt

TABLE XIII

OPERATIONS PERFORMED BY THE CHAR ROUTINE

1. Execute the SEI instruction to disable interrupts.
2. Send transmission enable signal to the Apple.
3. Wait for start bit from the Apple.
4. Retrieve 8 data bits, least significant bit first, from the Apple and combine into one data byte.
5. Send transmission disable signal to the Apple.
6. Execute the CLI instruction to enable interrupts.

processing of the keyboard and video interrupts. The CHAR routine initiates the transfer process by sending a high logic enable signal to the Apple II. This is accomplished by writing a one state to bit two of user port B at memory address location $DD01 hex. As discussed in Chapter II, writing to port B will set the specified voltage levels on all port B pins specified as outputs in the data direction register at memory location $DD03 hex. Reading port B will receive the logic state of all user port B pins specified by the data direction register as input pins.[14]

After a small delay following receipt of the high enable signal sent by the C-64 to the Apple II, the Apple II will send a start bit, a logic low, followed by eight

data bits upon receiving the enable signal from the C-64. As soon as the transmission enable signal is sent by the CHAR routine from the C-64, the CHAR routine immediately starts reading port B looking for the start bit being sent from the Apple II. The start bit will be received by the C-64 at the least significant bit (LSB) location of port B. When the least significant bit changes logic state and goes to a low logic state, the start bit has been received. This start bit synchronizes the data being sent from the Apple II with the read strobes of the C-64 so that the proper data is received. The time delay between data bits sent from the Apple II (discussed more fully later) is a constant 430 microseconds (us). Thus, the CHAR routine after receiving the start bit will delay approximately 215 us to wait for the middle of the start bit to pass. The CHAR routine will read port B every 430 us thereafter, until the eight data bits have been received from the Apple II. After the eight data bits have been sent, the Apple II sends two stop bits. The CHAR routine waits for two more 430 us time intervals then disables transmission by writing a low logic value to bit 0 of user port B at memory location $DD01 hex. The CHAR routine then returns control to the main transfer program. Thus, transmission is disabled until the CHAR routine is

again called by the main program. Figure 4 shows the timing waveforms on the C-64 user port B during the transfer of a single character, the letter A.

Although the time delay of 430 us was arrived at by trial and error, this value of time delay does have some reasonable explanations. The time delays generated by the Apple II and the C-64 are implemented by software delay loops. These delay loops have an integer number of instruction cycles. The Apple II delay loop is 440 instruction cycles. The C-64 delay loop is 430 instruction cycles. The time needed to implement an instruction cycle is determined by the main clock speed of the host computer. The main clock speed of the Apple II is 1.023 megahertz (MHZ).[15] The main clock speed of the C-64 is 1.000 MHZ.[16] These clock speeds determine the number of instruction cycles executed for any given specified period of time. For example, in one millisecond the Apple II will execute 1023 instruction cycles, however, the C-64 will execute only 1000 instruction cycles.

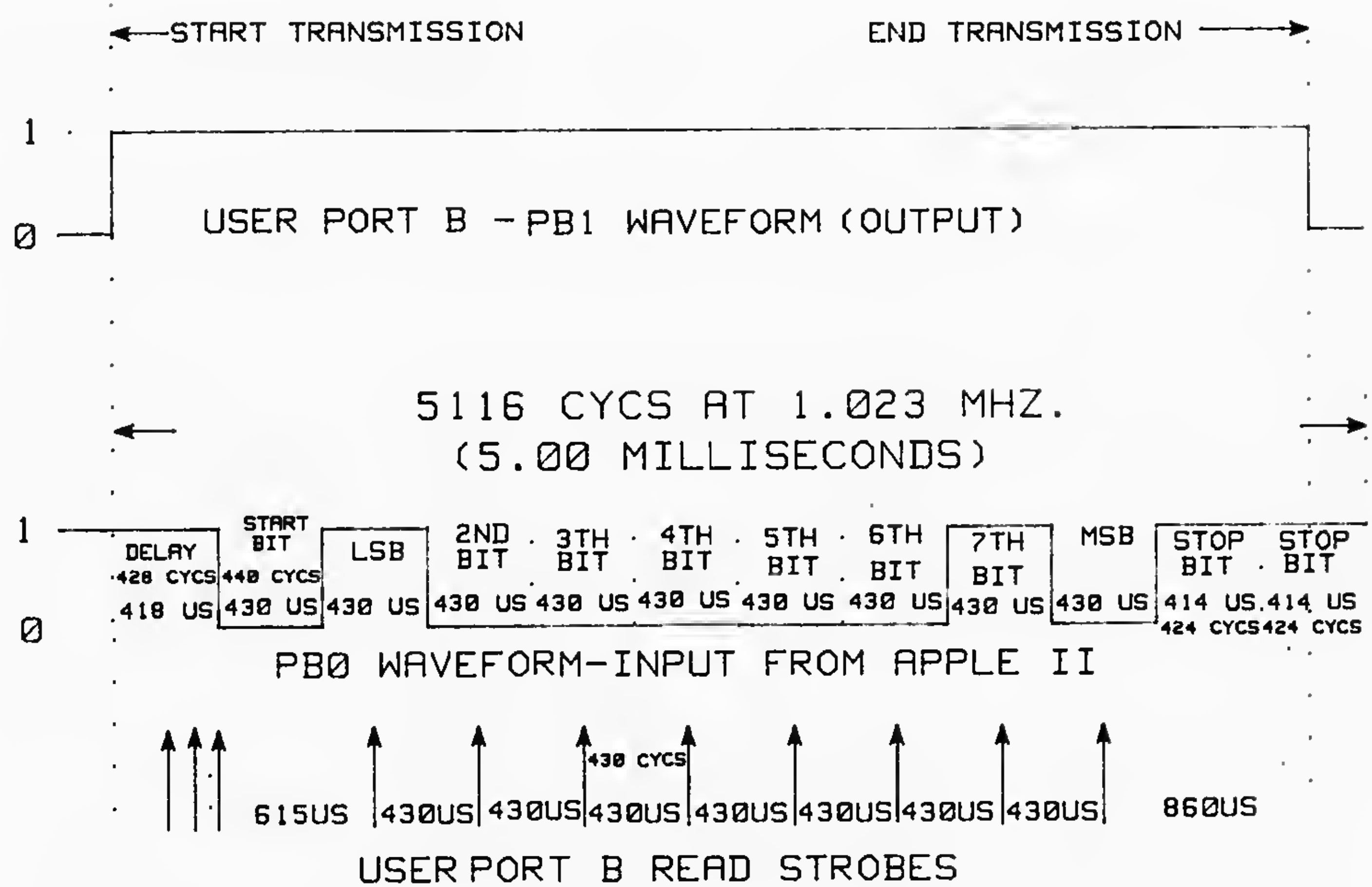

FIGURE 4 C-64 USER PORT WAVEFORMS FOR THE LETTER A

With the different clock rates, the problem that occurs in generating equal time delays is:
What integer number of clock cycles when executed on the Apple II will give a time delay which will allow an integer number of clock cycles to be executed on the C-64 in an equal amount of time and give a reasonable transmission speed?

One answer is 1023 instruction cycles on the Apple II and 1000 instruction cycles on the C-64 which will give time delays of 1 millisecond. Although this answer is acceptable, the transmission rate is a little slow (1000 BAUD). The trial and error solution of 440 Apple II instruction cycles and 430 C-64 instruction cycles gives almost equal time delays of 430 us with a difference error of only 0.025% (107 nanoseconds difference) with a transmission rate of 2200 BAUD. Although higher transmission rates can be obtained, the present transmission rate of 2200 BAUD is used to allow for slight variation, if any, between the different types of C-64 and Apple II compatible computers. Also at higher transmission speeds, interface cable length, environmental noise, no error checking protocol, and TTL level voltages being used, all increase the probability of a transmission error occurring.

2. Sequential File Transfer

After receiving the name of the file to be transferred and the sequential file type specifier, the C-64 transfer program jumps to the sequential file transfer section. The sequential file transfer section will OPEN a sequential file on the diskette with the name received from the Apple II. Then a data character is received from the Apple II using the CHAR routine and written to the open sequential file on the diskette. This process continues, a character at a time, until a zero is received from the Apple II. A zero designates the end of the sequential file. The sequential file is closed and control is returned to the Basic MENU program and another file transfer may be initiated.

Any drive errors (e.g. write protect error, device not found error) that may occur during the transfer will cause the transfer program to prematurely abort, close the sequential file, and return to the MENU program. However, the Apple II will seem to lock up, waiting to send another data character. Simultaneous pressing of the control and reset keys on the Apple II will be necessary to abort the transfer process and initiate another transfer.

3. Random Access File Transfer

After receiving the name of the file and the random access file type specifier from the Apple II, the

C-64 transfer program jumps to the random access file transfer section. The random access file transfer section receives the record length from the Apple II, the disk drive command channel is opened, and then a relative file is opened with the name and record length received from the Apple II. The random access file transfer program will receive the record number and the record position from the Apple II and transfer this information to the disk drive via the open command channel. Record data received from the Apple II is written to the open relative file on the diskette until a zero data value is encountered. A zero data value encountered signals the termination of the current record data. The random access file transfer program will receive the new record number and record position from the Apple II and transfer this to the disk drive via the command channel. The new record data is received from the Apple II and written to the relative file. This process will continue until four consecutive zero's are received from the Apple II. The four zeros signal the termination of the transfer of data from the Apple II. The relative file and the command channel is closed and control is returned to the Basic MENU program. Any drive errors that might occur during transfer will cause a premature abort of the transfer and

control will be returned to the MENU program. However, the Apple II will seem to lockup waiting to send another data character. Simultaneous pressing of the control and reset keys on the Apple II will be necessary to abort the transfer process so that another transfer can be initiated.

4. Basic File Transfer

The Basic MENU program allows the user to select a Basic file to be transferred. If a Basic file is selected the user may select the Apple character set option and or the Apple emulation option. Selecting the various options affects the transfer process. These transfer options and their affects on the transfer are described in further detail later in this chapter. After the user makes his selections the Basic MENU program jumps to the C-64 transfer program.

The C-64 transfer program will receive the Basic program file name and type specifier from the Apple II. Then the C-64 transfer program will jump to the Basic File transfer section of the C-64 transfer program.

The Basic file transfer section will store in the memory address location where Basic programs are stored the additional Basic program option lines selected by the user. These program lines will LOAD the user selected options when the transferred Basic program is run. See

Appendix A - The TransVersion User Guide for more details.

The Basic file transfer program will begin receiving the Apple Basic program data bytes from the Apple II. The Basic file transfer program will store directly in the C-64 memory all line number bytes, end of line pointer bytes, line link pointer bytes , and all data byte values less than 128 (non-tokens). All Apple data byte values greater than or equal to 128 (i.e. tokens) will be first converted to the equivalent C-64 data byte value (token) by use of conversion tables before being stored in memory. A token is a one byte code which represents a specific command in Basic (i.e. 143="REM", 128="END"). The Apple tokens which have an equivalent C-64 token (i.e. they represent the same Basic command) and their equivalent C-64 token values are listed in Table XIV.[17,18]

The conversion process is dependent on three factors which are listed in Table XV.

If the Apple token has an equivalent C-64 token then the value of the Apple token is used to point to the equivalent C-64 token value. The equivalent C-64 token is retrieved from the conversion table and stored in memory replacing the original Apple token value.

TABLE XIV

EQUIVALENT TOKEN VALUES FOR THE APPLE AND THE C-64

APPLE	ENGLISH	C-64	APPLE	ENGLISH	C-64
128	END	128	199	STEP	169
129	FOR	129	200	+	170
130	NEXT	130	201	-	171
131	DATA	131	202	*	172
132	INPUT	133	203	/	173
134	DIM	134	204	^	174
135	READ	135	205	AND	175
170	LET	136	206	OR	176
171	GOTO	137	207	>	177
172	RUN	138	208	=	178
173	IF	139	209	<	179
174	RESTORE	140	210	SGN	180
175	&	38	211	INT	181
176	GOSUB	141	212	ABS	182
177	RETURN	142	213	USR	183
178	REM	143	214	FRE	184
179	STOP	144	217	POS	185
180	ON	145	218	SQR	186
181	WAIT	146	219	RND	187
182	LOAD	147	220	LOG	188
183	SAVE	148	221	EXP	189
184	DEF	150	222	COS	190
186	PRINT	153	223	SIN	191
187	CONT	154	224	TAN	192
188	LIST	155	225	ATN	193
189	CLR	156	227	LEN	195
191	NEW	162	228	STR$	196
192	TAB(	163	229	VAL	197
193	TO	164	230	ASC	198
194	FN	165	231	CHR$	199
195	SPC(	166	232	LEFT$	200
196	THEN	167	233	RIGHT$	201
198	NOT	168	234	MID$	202

TABLE XV

THREE FACTORS THAT AFFECT THE CONVERSION OF TOKENS

1. What is the value of the data byte (token) received from the Apple?

2. Was the emulation option selected by the user?

3. Does the Apple token (data byte) have an equivalent value in Commodore Basic?

If the Apple token value has no equivalent token value (Ref. Table XIV, e.g. 137,144) in C-64 Basic then a check is made to see if the emulation option was selected. If the emulation option was selected and the value of the Apple token has an equivalent emulated C-64 token value (Ref. Table XV, e.g. 137,144) then this value is retrieved from the conversion tables and stored in the Basic program memory space. See Table XVI for a list of the emulated C-64 token values.

However, if the emulation option was not selected or the Apple token has no equivalent emulated C-64 token, then the Apple token value points to an ASCII data string which represents the English equivalent of the Apple Basic command the Apple token value represents (Ref. Table XVII and Table XVIII). This data string contains additional underline characters which are stored immediately before and after the ASCII command string to readily identify the

TABLE XVI

APPLE TOKEN VALUES VS EMULATED C-64 TOKEN VALUES

APPLE	ENGLISH	C-64	APPLE	ENGLISH	C-64
137	TEXT	208	158	INVERSE	210
144	HGR2	211,50	159	FLASH	209
145	HGR	211	161	POP	213
146	HCOLOR=	220,176,178	162	VTAB	207
147	HPLOT	212	163	HIMEM:	219,58
150	HTAB	204	164	LOMEM:	218,58
151	HOME	205	169	SPEED=	216,178
155	TRACE	206	190	GET(GIT)	214
156	NOTRACE	168,215	216	PDL	226
157	NORMAL	217,167,217			

Apple command that is unavailable on the C-64. See Table XVII and Table XVIII for the Apple command strings that are used for unavailable or incompatible Apple commands on the C-64.

A REM code (No. 143) is also stored at the beginning of the line to comment out the line containing the unavailable Apple command. The conversion process continues until three consecutive zeros are received from the Apple II. The three zeros signal the end of the transfer of the data from the Apple II.

After the transfer of data from the Apple II the transfer program must convert the Apple disk commands. The Apple disk commands, unlike the C-64 disk commands, are untokenized ASCII data strings embedded within PRINT command statements, thus the disk commands are not

TABLE XVII

C-64 COMMAND STRINGS FOR INCOMPATIBLE APPLE COMMANDS

APPLE TOKEN VALUE	ENGLISH VERSION
133	_DEL_
136	_GR_
138	_PR#_
139	_IN#_
140	_CALL_
141	_PLOT_
142	_HLIN_
143	_VLIN_
148	_DRAW_
149	_XDRAW_
152	_ROT=_
153	_SCALE=_
154	_SHLOAD_
160	_COLOR=_
165	_ONERR_
166	_RESUME_
167	_RECALL_
168	_STORE_
185	_POKE_
197	_AT_
215	_SCRN(_
226	_PEEK_

converted during the transfer process and must be converted after the transfer from the Apple II.

The Basic file transfer program uses a table look up procedure to find all the Apple disk commands. The Apple disk command, once found, will be replaced by the equivalent C-64 disk command, if available. Otherwise the line containing the unavailable disk command is commented (REM) out. See Table VI of Appendix A for the list of Apple disk commands converted to C-64 syntax.

TABLE XVIII

C-64 COMMAND STRINGS IF EMULATION OPTION IS NOT SELECTED

APPLE TOKEN VALUE	ENGLISH VERSION
137	_TEXT_
144	_HGR2_
145	_HGR_
146	_HCOLOR=_
147	_HPLOT_
150	_HTAB_
151	_HOME_
155	_TRACE_
156	_NOTRACE_
157	_NORMAL_
158	_INVERSE_
159	_FLASH_
161	_POP_
162	_VTAB_
163	_HIMEM:_
164	_LOMEM:_
169	_SPEED=_
190	_GET_
216	_PDL_

See Table VII of Appendix A for the Apple commands not converted to C-64 syntax.

If the Apple disk command has no equivalent on the C-64, a check is made to see if the emulation option was selected by the user. If the emulation option was selected by the user, then a check is made to determine if the Apple disk command is a C-64 emulated disk command. If the Apple disk command is an emulated disk command, the proper C-64 emulated disk command is substituted for the Apple disk command. See Table VI of Appendix A for the

Apple disk commands that are emulated on the C-64.

After all the Apple disk commands have been converted to the equivalent C-64 disk commands, the Basic transfer program will convert all C-64 PRINT commands that follow the original Apple WRITE command to C-64 PRINT# commands to correct the difference in the way the Apple Basic and the C-64 Basic write data to a disk file. The conversion of the PRINT commands to PRINT# commands will cease if an Apple CLOSE disk command is encountered, because output is once again directed to the screen. Likewise all the original Apple INPUT and Apple GET commands will be converted to C-64 INPUT# and GET# after an Apple READ disk command until an Apple CLOSE disk command is encountered. See Appendix A - Disk Command Translation for further details for the reasons why this is necessary.

After all the disk command conversion processes are accomplished by the Basic Transfer program, the corrections for the length of the Basic program lines occurs. This is done because Applesoft Basic will allow the editing of up to 255 characters in one line, while C-64 Basic will allow only 80 characters per line to be edited. Nevertheless, C-64 Basic will RUN lines up to 255 characters.[19]

The need to split a Basic program line is

determined by adding the length value of all items of the line that is displayed when editing or LISTing of the line. The length, the number of characters of the line number, is based on the value of the line number. The length of the line number is one for line numbers less than ten, two for line numbers between 10 and 99, and three for line numbers between 100 and 999 etc. All ASCII data values have a length of one and tokens have lengths corresponding to the number of characters in the Basic command they represent (i.e. 143 = "REM" = length of 3). As the transfer routine adds up the number of characters per line using the length tables, the transfer routine determines where in the line the line can be split. If the Basic program line needs to be split the transfer routine will split the line into two lines creating a new line number. Some Basic lines cannot be split, e.g. If statements; such statements are left unchanged. The transfer routine splits the Basic program line based on the rules described in Appendix A - Splitting Long Program Lines.

After all the Basic program lines have been through the length processing, the Basic program is stored on the diskette using the file name received from the Apple II. After the transferred Basic program is stored

on diskette, the Basic MENU program is LOADed into memory. The C-64 transfer program returns control to the MENU rogram and another file transfer may be initiated.

C. The C-64 Apple Emulation Programs

Emulation of Apple commands and statements on the C-64 requires an extensive knowledge of the C-64 operating system. This knowledge was gained primarily through reference to Mapping the Commodore 64 and The Anatomy of the Commodore 64 as shown in the bibliography. Then the concepts of these books were applied to modify ROM Kernel routines to meet the requirements of the emulation commands.

1. The Apple Emulation Program

The C-64 Apple emulation program adds new Basic keywords to the existing C-64 Basic command set. The new C-64 Basic keywords are interpreted and operate in a way similar to the original C-64 Basic keywords. The new keywords added to the C-64 Basic command set are listed in Table XIX.

The Apple commands listed in Table XIX was emulated on the C-64 (e.g. FLASH, HPLOT) to help reduce the number of Applesoft Basic commands not available on the C-64. The Apple commands that were emulated on the C-64 was chosen because of their frequency of use in Apple

programs, the degree of difficulty in creating equivalent C-64 Basic subroutines that do the same function, and to

TABLE XIX

APPLE KEYWORDS ADDED TO THE C-64 BASIC COMMAND SET

TOKEN NUMBER(S)	KEYWORD	TOKEN NUMBER(S)	KEYWORD
204	HTAB	217,176,217	NORMAL
205	HOME	218,58	LOMEM:
206	TRACE	219,58	HIMEM:
207	VTAB	220,176,178	HCOLOR=
208	TEXT	221	EXEC
209	FLASH	222,147	BLOAD
210	INVERSE	222,148	BSAVE
211	HGR	222,138	BRUN
212	HPLOT	223,188	CATALOG
213	POP	224	PAUSE
214	GIT(GET)	225	KILL
168,215	NOTRACE	226	PDL
216,178	SPEED=		

simplify the conversion process during transfer of Applesoft Basic programs containing Apple commands with unavailable C-64 equivalent commands.

The Applesoft Basic commands not emulated on the C-64 (See Table XIX of Appendix A - The TransVersion User Guide) were not supported by the emulation software because of the emulation program development time constraints, limited available RAM memory on the C-64, and the low frequency of use of the commands in Applesoft Basic programs. Although the Applesoft commands not

supported by the emulation program limits the usefulness of the TransVersion system it is not a serious limitation because of the limited use of these commands in Applesoft Basic programs.

The C-64 emulation program works in conjunction with the C-64 Basic operation system. The C-64 emulation program is inserted into the Basic operating system via the Basic indirect vector table located in RAM. This table, which starts at location 768 ($300 hex) and continues to 819 ($333 hex), contains two byte address vectors of the various routines needed by the C-64 Basic operating system. When the C-64 Basic operating system wants to execute one of these routines, it will reach the routine needed via the Basic indirect vector table. At power-on time, these vectors are set to point to the normal ROM Basic routines.[20] However, by changing these vectors the C-64 emulation program can modify the routines needed to emulate the new Basic keywords listed in Table XIX. The indirect vectors changed by the C-64 emulation program are listed in Table XX.[21]

In order for the C-64 emulation program to add new keywords to the Basic operating system, and to be able to LIST and RUN them, the C-64 emulation program must intercept the Basic operating system routines that

tokenize, detokenize, and execute the keyword tokens.[22] The C-64 emulation program contains the various routines needed to add the new keywords. The C-64 emulation program adds these new routines by changing the indirect

TABLE XX

INDIRECT VECTORS CHANGED BY THE C-64 EMULATION PROGRAM

RAM ADDRESS	NAME	FUNCTION
$300	IERROR	PRINT BASIC ERROR MESSAGE
$302	IMAIN	MAIN BASIC INPUT PROGRAM LOOP
$304	ICRNCH	TOKENIZES KEYWORDS INTO TOKENS
$306	IQPLOP	LIST BASIC PROGRAM TOKENS AS TEXT
$308	IGONE	EXECUTES BASIC TOKENS
$30A	IEVAL	EXECUTES BASIC FUNCTIONS
$314	CINV	IRQ INTERRUPT ROUTINE
$316	CBINV	BRK INTERRUPT ROUTINE
$318	NMINV	NMI INTERRUPT ROUTINE
$324	IBASIN	GET A CHARACTER
$326	IBSOUT	OUTPUT A CHARACTER
$328	ISTOP	STOP KEY ROUTINE
$32A	IGETIN	GET ONE BYTE FROM INPUT DEVICE
$32C	ICLALL	CLOSE ALL FILES
$330	ILOAD	LOAD RAM FROM A DEVICE

vector locations listed in Table XX. The Basic operating system will then gain access to the C-64 emulation program routines. The C-64 emulation program when installed in the C-64 RAM memory will change the Basic indirect vectors listed in Table XX to point to the new routines in the C-64 emulation program.

The operation of the various C-64 emulation

routines closely parallels the operation of the C-64 basic operating system. A description of the C-64 emulation routines will describe in detail the operation of the C-64 emulation program. The following description of the C-64 emulation program will be done by describing each individual routine pointed to by the indirect vector locations listed in Table XX.[23]

a. The Error Handling Routine - ERRHND The IERROR vector at memory address location $300 hex is changed to point to the ERRHND routine at memory address location $98E5 hex. The IERROR vector is used by the Basic operating system as a dual purpose handler, as a routine to print an error message or the READY message. The ERRHND routine checks if an error occurred, if an error occurred, an error beep is sounded, all EXEC flags are cleared, any EXEC file is closed, all I/O devices are set to default, the normal TEXT screen is displayed, and then program control is returned to the Basic ROM error handler. The ERRHND routine thus resets all default conditions that might have been changed during graphics or EXEC file operations before the error message is displayed by the C-64 Basic operating system.

b. The Main Input Routine - MAINA The IMAIN vector at memory address location $302 hex is changed to

point to the MAINA routine at memory address location $9AE9 hex. The IMAIN vector is used by the Basic operating system to point to the main input loop used when in direct mode. The main input loop is used to execute statements or store Basic program lines into memory. The main input loop routine determines if a input statement line get executed or stored into memory by checking the begin of the statement for a line number. If a line number exist then the input statement is stored into memory as a Basic program line, otherwise the input statement is executed. If a Basic program is running then the main input loop is used to execute the Basic statements in the program. The MAINA routine checks to see if an EXEC file is open. If an EXEC file is open then control is diverted to the appropriate EXEC routines to allow the EXEC files to control operation, otherwise, program control is returned to the Basic main input loop. The MAINA routine is needed because all direct mode commands executed by the EXEC file return to the main input loop after execution of the command and the MAINA routine must divert control back to the appropriate EXEC command routines after execution of the direct mode command.

c. <u>The Tokenization Routine - TOKNIZ</u> The ICRNCH vector at memory address location $304 hex is changed to

point to the TOKNIZ routine at memory address location $C009 hex. The ICRNCH vector is used by the Basic operating system to point to the CRUNCH routine which is used to tokenize the C-64 Basic keywords. The TOKNIZ routine first calls the CRUNCH routine which is used to tokenize all normal keywords. The tokenization process is somewhat tricky, in that new keywords cannot be tokenized if they are included in DATA statements, REM statements, or included as a literal string in quotes. The TOKNIZ routine will tokenize all the new keywords listed in Table XVI to their equivalent token(s) as listed in Table XVI. The TOKNIZ routine uses a table look up procedure to identify the new Basic keyword and its token value. The last letter of each keyword in the table is identified by setting the MSB to a one. The end of the table is marked by a zero. The input data is compared to the data in the keyword table, if a match is found the token is retrieved from the table, and substituted for the keyword and the token is stored in memory, otherwise the original data is stored in memory. This process continues until the complete input data line is tokenized. The TOKNIZ routine handles all the special case keywords that have embedded keywords. An embedded keyword is a keyword contained within a keyword (e.g. HCOLOR=, BSAVE). The tokenization

process is used to decrease the amount of RAM memory a program needs when stored in the computer.

d. <u>The Detokenization Routine - PRTOK</u> The IQPLOP vector at memory address location $306 hex is changed to point to the PRTOK routine at memory address location $C0EA hex. The IQPLOP vector is used by the Basic operating system to point to the QPLOP routine which prints Basic tokens as ASCII text characters of their respective keyword. The PRTOK routine checks if the token is a new token value. If the token is a new token value the PRTOK routine uses a table look up procedure to print the keyword represented by the token, otherwise if the token is not a new token then the PRTOK routine returns control to the original Basic routine QPLOP to print the Basic token's keyword.

e. <u>The Execute Statement Routine - EXEST</u> The IGONE vector at memory address location $308 hex is changed to point to the EXEST routine at memory address location $C171 hex. The IGONE vector is used by the Basic operating system to point to the GONE routine which gets the next token and executes the token. The EXEST routine determines if the TRACE command is active. If the TRACE command is active then the current line number is displayed in brackets. The EXEST routine then gets the next token and determines if the token is a new token. If

the token is a new token, the new token is executed by the EXEST routine. Otherwise, if the token is an original token value then program control is return to the Basic GONE routine. Selected original tokens (i.e. IF, PRINT) are executed with the EXEST routines because they must be modified to function properly with the new tokens.

The value of the new token is used to locate the subroutine that performs the function represented by the token. The proper routine is located by the EXEST routine by subtracting 204 from the token, multiplying by two, and then retrieving the proper address using the resultant value as a pointer of a memory address look up table. This address is pushed on the stack and a return (RTS) is executed. This procedure causes the program to jump to the subroutine which will be executed to perform the function that the new token represents. The operation of the 25 individual subroutines that perform the keyword functions will not be described here. See Appendix D for the commented source listings of the keyword function subroutines for further details.

f. <u>The Execute Function Routine - EXEFUN</u> The IEVAL vector at memory address location $30A hex points to the EXEFUN routine at memory address location $C240 hex. The IEVAL vector is used by the Basic operating system to

point to the EVAL routine which is used to evaluate functions (i.e. INT, ABS, etc.). The EXEFUN routine checks to see if the token is a new function token (i.e. PDL). If the token is a new function token, the token value is used to pickup the function routine address similar to the EXEST routine and the function is executed; otherwise if the function token is an original C-64 Basic function token then control is returned to the Basic EVAL routine.

g. <u>The IRQ Interrupt Routine - IRQRPT</u> The CINV vector at memory address location $314 hex is changed to point to the IRQRPT routine at memory address location $9B2D hex. The CINV vector is used by the Basic operating system to point to the IRQ routine which handles all IRQ interrupts. The IRQ routine updates the software clock, checks the stop key, blinks the cursor, and reads the keyboard. The IRQRPT routine adds a raster interrupt routine for the HGR mode which uses the IRQ vector to split the screen display. The IRQRPT routine also adds a blinking character routine used during Flash mode to cause the Flash characters to change at a steady rate from NORMAL text to INVERSE text and vice versa. During HGR mode the raster interrupt of the VIC II chip is active. The VIC II chip interrupts at raster scan lines 217 or 250.

The IRQRPT routine will determine at which scan line the interrupt was generated, sets up the graphics or text display, and sets up the VIC II chip to interrupt at the alternate raster scan line interrupt. The normal timer interrupt (CIA #1 Timer B) is checked to see if the CIA #1 chip needs service, if the service is needed then control is returned to the normal IRQ routine to handle the normal interrupt processes. Otherwise, if the CIA #1 chip does not need service the FLASH characters are toggled if it is time to blink the FLASH characters then control is returned to the interrupted program. If the HGR mode is not active then the normal timer (CIA #1 Timer B) caused the IRQ interrupt. In this instance, the IRQRPT routine determines if the FLASH characters need to be blinked. If the FLASH characters do not need to be blinked then control is returned to the normal ROM IRQ routine. If the FLASH characters need to be blinked the FLASH characters are toggled to their alternate state and control is returned to the ROM IRQ routine.

The IRQRPT routine determines what FLASH characters are to be toggled by checking a FLASH memory array for set bits (1). The FLASH memory array consists of 125 bytes (1000 bits) which corresponds to the 1000 text screen locations each bit in the memory array

corresponding to an individual location on the text screen. The IRQRPT routine scans the FLASH memory array for non-zero bits and toggles the corresponding text screen memory location by doing an EOR #$80 with the data in the selected text screen location and storing the converted data into the same memory location. This causes the character to appear to flash, that is, it blinks from a normal character to an inverse character and vice versa. The IRQRPT routine does not update the FLASH memory array to keep track of the location of the FLASH characters. The updating of the FLASH memory array is handled by the INPUT and OUTPUT routines described later.

h. <u>The Run-Stop/Restore Routine - NEWRSR</u> The CBINV vector at memory address location $316 hex is changed to point to the NEWRSR routine at memory address location $9860 hex. The CBINV vector is used by the Basic operating system to point to the BRK routine which is used to handle the BRK instruction when it is encountered by the 6510 CPU. Also, the BRK routine is executed when the run-stop/restore keys are pressed simultaneously. The BRK routine will initialize the VIC chip, the CIA chips, the SID chip, restore the indirect vectors in the lower RAM indirect vector table and jump to the Basic warm start vector at $A002. The Basic warm start routine closes all files, sets the default devices, resets the stack and

Basic program pointers, and jumps to the Main input (READY) loop. The NEWRSR routine does the same functions as the BRK routine, however, before jumping to the Basic warm start vector the C-64 emulation program is checked for intactness by doing a checksum on the program. If the emulation program is intact the emulation program is executed, a beep is sounded, and the warm start routine is executed. Otherwise an error message is printed and a normal warm start is executed.

i. <u>The Restore Key Routine - RWDG</u> The NMINV vector at memory address location $318 is changed to point to the RWDG routine at memory address location $984C hex. The NMINV vector is used by the Basic operating system when a non - maskable interrupt (NMI) occurs. The NMI will occur when the restore key is pressed or when CIA #2 interrupts the CPU during RS232 operations. If the restore key is pressed simultaneously with the stop key then the BRK routine is entered. The RWDG routine alters this sequence, if the restore / run-stop keys are pressed then program control is returned to the NEWRSR routine instead of the BRK routine. If restore key alone is pressed then only a beep is sounded and control is returned to the interrupted program. However, if the interrupt is caused by the CIA #2 chip during RS232

operations then the RS232 interrupt is ignored and the restore key is assumed to be the cause of the interrupt. Thus the RWDG routine disables all RS232 type operations.

j. The Input Routine - INPUT The IBASIN vector at memory address location $324 hex is changed to point to the INPUT routine at memory address location $9BEF hex. The IBASIN vector is used by the Basic operating system to input a character or characters from the current input device. The IBASIN vector normally points to the CHRIN routine. The CHRIN routine echoes to the screen all characters retrieved from the keyboard. The INPUT routine intercepts the CHRIN routine at this point to check the keyboard character value before displaying to the screen. This allow the INPUT routine to update the FLASH memory array used by the IRQRPT routine during FLASH mode. The INPUT routine determines the location on the screen at which the character is to be displayed. The INPUT routine also checks the value of the character to be displayed for one of the screen control characters or cursor control characters (i.e. clr key, down arrow key). The INPUT routine will determine if the key character to be displayed, due to its location or its value, will clear the screen or scroll the screen up one line. If either will occur the INPUT routine will update the FLASH memory array. The INPUT routine updates the FLASH memory array

by zeroing all bytes (clearing screen) or scrolling up one line line by moving all bytes in the FLASH memory array by five bytes (one row) and zeroing the last five bytes.

k. <u>The Output Routine - PRINT</u> The IBSOUT vector at memory address location $326 hex is changed to point to the PRINT routine at memory address location $C272 hex. The IBSOUT vector is used by the Basic operating system to output a character to the current output device. The PRINT routine determines if the Basic PRINT command is active. If the Basic PRINT command is active, the PRINT routine will delay printing of the character if the SPEED= command is active. Then the PRINT routine will determine if the INVERSE command is active. If the INVERSE command is active and the character to be output is a $0D hex (a return character), then the character is displayed followed by $12 hex to reestablish INVERSE mode. The INVERSE mode is entered by sending a $12 hex character and disabled by sending a $0D hex character. In all instances, The PRINT routine will sound a beep (bell) if a character code of seven is to be printed. Also, the PRINT routine will update the FLASH memory array in the same way as the INPUT routine before sending the character to the screen. The PRINT routine will then output the character to the screen and return control to Basic ROM.

l. <u>The Stop Key Routine - STPKEY</u> The ISTOP vector at memory address location $328 hex is changed to point to the STPKEY routine at memory address location $99DB hex. The ISTOP vector is used by the Basic operating system to point to the STOP routine. The STOP routine resets the default devices (keyboard and screen) if the stop key is pressed. The STPKEY routine resets the default devices, resets all EXEC flags and closes the EXEC file before returning control to Basic ROM. The STPKEY routine allows the operator to regain control from the EXEC file by pressing the stop key.

m. <u>The Get Routine - GETINA</u> The IGETIN vector at memory address location $32A hex is changed to point to the GETINA routine at memory address location $99FB hex. The IGETIN vector is used by the Basic operating system to point to the GETIN routine. The GETIN routine is used to retrieve one character from the current input device. The GETINA routine will determine if the APPLE GET (GIT) command is active. If the GIT command is active the GETINA routine will call the GETIN routine until a key is pressed (a non-zero value). The GETINA routine corrects the major difference between the C-64 Basic GET command and the Applesoft GET command. The Applesoft GET command will wait for a key stroke before continuing to the next command, where as the C-64 GET command will go to next

command even if no keystroke occurs.

n. The Close All Files Routine - CLRALL The ICLALL vector at memory address location $32C hex is changed to point to the CLRALL routine at memory address location $9ADA hex. The ICLALL vector is used by the Basic operating system to point to the CLALL routine. The CLALL routine is used to close all open files. The CLRALL routine determines if the EXEC command is active. If the EXEC command is active, then the CLALL routine is bypassed and control is returned to Basic ROM. If the EXEC command is not active then control is returned to the CLALL routine. The CLRALL routine allows the EXEC file to stay open during the CLR command and keeps the EXEC file in control of computer operations during execution of the EXEC file.

o. The Load File Routine - ALOAD The ILOAD vector at memory address location $330 hex is changed to point to the ALOAD routine at memory address location $C2C8 hex. The ILOAD vector is used by the Basic operating system to point to the LOAD routine. The LOAD routine is used to LOAD programs into RAM memory. The ALOAD routine intercepts the LOAD routine to pick up the starting address of a binary PRG files for the BRUN command. The ALOAD routine stores the saved start address

in a reserved memory location. The ALOAD routine then returns control to the Basic ROM routine LOAD. The BRUN command will use the reserved memory location to indirectly jump to the binary program just loaded to execute the binary program.

p. <u>The Emulation Program Enable Routine - INSTAL</u> The INSTAL routine is used by the C-64 Basic MENU program to attach the C-64 Apple Emulation program to the C-64 Basic operating system. The INSTAL routine will first determine if the C-64 Apple Emulation program is intact. If the C-64 Apple Emulation program is not intact then an error message is printed and installation of the C-64 Apple Emulation program is aborted. If the C-64 Apple Emulation program is intact and already installed then the KILL command routine is executed and the C-64 Apple Emulation program is then reinstalled. Installation of the C-64 Apple Emulation program by the INSTAL routine is accomplished by changing the indirect vector locations listed in Table XX. The INSTAL routine also determines if the Apple character set is needed and changes the TEXT screen location if necessary. The INSTAL routine will change the color memory location codes to the color black for the TEXT screen. The message 'APPLE' is displayed on the TEXT screen to inform the user the C-64 Apple

Emulation program is successfully installed.

q. <u>The Emulation Program Disable Routine - KILL</u> The KILL routine is used by the KILL command to disable the C-64 Apple Emulation program. The KILL routine restores all the indirect vector locations to the default values. The KILL routine resets the TEXT screen to its default location. The KILL routine closes any EXEC file that might be open. To enable the C-64 Apple Emulation program after a KILL command the INSTALL routine must be run by executing the C-64 Basic command SYS 49152.

The seventeen routines above describe the interface between the C-64 Apple Emulation program and the C-64 Basic operating system. Although the 25 individual C-64 Apple Emulation command routines (i.e. TEXT, HTAB, GIT etc.) were not described. The commented source listings in Appendix D adequately describe the operations of the C-64 Apple Emulation command routines.

2. <u>The Apple Character Set Program</u>

The Apple Character Set program will generate the Apple character set data in RAM to be used by the VIC chip. The Apple Character Set program will allow the C-64 to display the Apple character set, instead of the C-64 character set. The Apple character set is stored under ROM in RAM memory at memory address locations $A000 hex to $BFFF hex. The Apple Character Set program retrieves the

selected data for the new character set from the C-64 character ROM and written to the RAM at the new character set location in RAM. Selected RAM locations are changed to coincide with the Apple character shapes needed that are not normally available with C-64 Basic operating system. A reserved memory location is set to allow the C-64 Apple Emulation program to install the new character set when the C-64 Apple Emulation program is installed.

The fundamental difference between the Apple character set and the C-64's two character sets is that the Apple character set has both upper and lower case letters whereas the C-64 character sets have either upper case or lower case letters, but not both. The Commodore key can toggle either character set active at any time.

The capability to have upper and lower case letters active at all times allows the C-64 to emulate the Apple II character set. The Apple Character Set program will give the C-64 computer this capability.

IV. IMPLEMENTATION

A. Installation Of The TransVersion System Software

A primary consideration in designing the TransVersion system software is to keep simple the installation and use of the software. Apple Computer Inc. has designed in the Apple disk operating system one of the simplest application software installation procedures available. The Apple disk operating system during the power up sequence will LOAD and RUN an application specific Basic program from the diskette. This start up program, usually named HELLO, will start execution of the specific application software.[24]

The HELLO program in the TransVersion system controls the software loading procedure as requested by the user. The HELLO program prompts the user for selected information needed for transferring of the selected file to the C-64, LOADs the appropriate TransVersion system software, and monitors the transfer process for successful completion of the transferred file. The HELLO program allows the transfer process to be a user friendly procedure. The installation and operation of the Apple TransVersion software is further described in Appendix A - The TransVersion User Guide.

Unlike the Apple disk operating system, the C-64

operating system does not automatically LOAD and RUN a start up program during the power up sequence. The LOADing of the application specific software is done by the user. The TransVersion system user must LOAD and RUN the Basic MENU program from the TransVersion diskette. The C-64 Basic MENU program, similar to the Apple HELLO program, will control the LOADing of the various application specific programs specified by the user. The C-64 MENU program allows the user to select either of two modes, emulation mode or transfer mode. Selection of the emulation mode by the user will cause the LOADing of the C-64 Apple emulation program by the MENU program. The user will then execute all subsequent operations under the emulation mode. The various user operations affected by the emulation program are the RUNning of a Basic program, LISTing of a Basic program, and the editing of a Basic program. These operations modified by the emulation program allow the user to debug the Basic programs transferred from the Apple computer which may have emulated commands embedded within them.

Selection of the transfer mode option by the user will initiate the transfer process from the Apple to the C-64. The C-64 Basic MENU program will LOAD the transfer programs from the TransVersion diskette, turn over control

of the computer to the transfer program during transfer of the specified program from the Apple, and receive control of the C-64 after the transfer process is complete. After transfer the MENU program will again request from the user what operating mode the user wishes to enter. The installation and operation of the TransVersion system is further described in Appendix A - The TransVersion User Guide.

B. Testing The Transversion System Software

1. Testing the Transfer Software.

The testing of the TransVersion system software and hardware was performed throughout the development of the software. The development and testing of the TransVersion system software was accomplished as listed in Table XXI.

During the development of the TransVersion system software, testing of the software was done throughout the development process as a debugging tool to correct errors that occurred during the software design stage. The testing procedure used throughout the design is develop the software, test the software, and modify software if the test results were unacceptable. The test procedure that was used to test the software was determined in part by the specific function of the software being tested.

TABLE XXI

TEST AND DEVELOPMENT SEQUENCE FOR THE TRANSFER SOFTWARE

1. Development and testing of the "send character" and "receive character" routines, SEND and CHAR, on the Apple and C-64. Note: This will test the interface cable hardware.

2. Development and testing of the transfer and conversion software for Applesoft Basic programs.

3. Development and testing of the C-64's Apple emulation software.

4. Development and testing of the Apple disk commands to the C-64 disk commands conversion software.

5. Testing compatibility of the transfer software and emulation software.

6. Development and testing of the transfer software for all type files other than Apple Basic program files (i.e. text, random access, and binary files).

7. Development and testing of the Basic driver programs, HELLO and MENU.

8. Final testing of the TransVersion software.

For example, in testing the send character (SEND) and receive character (CHAR) routines, a single character, the letter A for example, was transferred from the Apple to the C-64 thousands of times and checked if an error occurred during transfer (i.e. a letter A was not received). By testing the TransVersion system software with a sum of individual test routines, system development

occurred rapidly and minimized debugging time.

Testing of more complicated routines was done similarly. For example in testing the software for the ability to transfer files other than Basic program type files, complete files were transferred. The received file was visually checked against the Apple source file for transfer errors. This test procedure insures the correct operation of the transfer software during the transfer of sequential, random access and binary type files.

The testing of Applesoft Basic file transfer software was separated into various parts because of the complexity of the transfer and conversion process. Table XXII lists the separate test steps.

TABLE XXII

TEST STEPS ON SOFTWARE OPERATIONS DURING BASIC FILE TRANSFERS

1. Test of Basic program transfer software with no conversions.

2. Test of the Applesoft tokens to C-64 token conversion software.

3. Test of the Apple disk command syntax to the C-64 disk command syntax software.

4. Test of the line length conversion software.

5. Test of the various transfer options implementation software selected by the user (i.e Emulation option and Character Set option).

In the initial development of the transfer software for Applesoft Basic programs, the transfer software did not convert any of the Apple tokens to the C-64 equivalent tokens. This transfer process transferred the Apple program from the Apple to the C-64 directly. This procedure verified that the Apple program was stored properly in the C-64 RAM memory. The line numbers, the end of line pointers, and other Basic pointers were preserved and the transfer was completed properly.

After the simplest Basic file transfer procedure was completed and tested, then the token conversion process was introduced into the transfer procedure. The token conversion routine was tested by transferring an Apple Basic program file containing all of the Applesoft keywords (token values) to the C-64. The resultant received program in the C-64 was then compared with the Apple source program to verify that the conversion process was successfully completed. This test procedure was repeated many times throughout the development of the TransVersion system software to insure accurate results.

The next step in the development of the TransVersion system software was the development of the Apple disk command to C-64 disk command conversion software. The complexity of the Applesoft disk command to

the C-64 disk command conversion process requires that the process occurs after the transfer of the Basic program to the C-64. The testing of the disk command conversion process was accomplished by transferring an Apple program with many disk command to the C-64. The resultant transferred program was compared with the original Apple program for accuracy in syntax correction and disk command conversions. Tables VIII through XII of Appendix A - The Transversion User Guide illustrate samples of the type of test files used to test the disk command conversion process.

The line length conversion is accomplished after the disk command conversion. The line length conversion process test procedure is similar to the disk command process test procedure. The source test program to be transferred from the Apple contains many very long lines. Tables XIV and XV of Appendix A - The Transversion User Guide illustrate the original test program and the resultant transferred program, respectively.

The test procedure of the TransVersion system software Basic file transfer capability would not be complete without a final test. The final test was the transferring of a program, written for the Apple computer, to the C-64 then running the transferred program on the C-64. The Apple program RENUMBER INSTRUCTION which is

located on the DOS 3.3 System Master Disk was chosen as a test file, because this program contains no incompatible Apple soft Basic commands (e.g GR, FP, POKE), contains many Applesoft Basic commands emulated on the C-64 (e.g. HTAB VTAB), and is an Applesoft Basic program of medium length (2200 bytes). The transferred RENUMBER INSTRUCTION program operated as expected on the C-64.

The length of the Applesoft RENUMBER INSTRUCTION program is 2200 bytes. The actual transfer time was measured at 12 seconds. This time corresponds to a baud rate of 2016 bits per second (11 bits per byte). This rate is close to the theoretical maximum of 2200 baud. The delays of the program due to overhead processing account for the differences in the actual data rate and the theoretical value. The post transfer processing (conversion) of the transferred program took an additional time of approximately 48 seconds. These results show that the TransVersion system is capable of transferring an converting Applesoft Basic programs to the C-64 at quite acceptable speeds. The manual entry of a program the size of the Applesoft RENUMBER INSTRUCTION program would take several hours. The creating of the C-64 Basic routines to simulate the Applesoft Basic commands HTAB and VTAB commands might take several more hours. Thus the

TransVersion system speeds up the transfer and conversion of Applesoft Basic Programs by a factor of better than 100 to 1.

Many other Applesoft programs were created and transferred to the C-64 to test all the options allowed by the TransVersion system software.

The testing of the Basic driver programs MENU and HELLO was done near the completion of the development of the TransVersion system software package. All the various options of the Basic driver program were selected systematically to verify the proper mode of operation occurred. Also, various user input errors that might occur such as the wrong file type selected, a non existent file name selected, or the wrong diskette inserted, were simulated to verify that the Basic driver programs could handle all common errors accurately.

2. Testing of the Emulation Mode software

The development and testing of the emulation program occurred in several well defined steps as shown in Table XXIII.

The development of the emulation program to add the new Basic keyword to the C-64 operating system was done in a modular format. The emulation program was developed using several subroutines, described in Chapter III, working together to create the necessary operating

TABLE XXIII

STEPS IN THE DEVELOPMENT OF THE

EMULATION PROGRAM SOFTWARE

1. Development of a skeleton program to add new Basic keywords to the C-64 operating system.

2. Development and testing of the new Basic keyword subroutine which is then added to the skeleton program.

3. Modification of the new Basic keyword subroutine and/or the skeleton program, if needed.

4. Repeating of steps 2 and 3 until all the new Basic keywords were added to the skeleton program.

environment. The subroutines described in Chapter III make up a skeleton program from which the complete body of the emulation program can be built. The initial skeleton program contained routines to tokenize the new Basic keywords into tokens, detokenize the new tokens to ASCII text, execute the new tokens (i.e. new Basic keywords), and two subroutines that connect and disconnect the skeleton emulation program from the C-64 Basic operating system.

The skeleton emulation program was tested by installing the emulation program using the connecting subroutine INSTAL, the Basic command SYS 49152 would accomplish this installation by executing the INSTAL

Routine. The test procedure using the new Basic keyword KILL in a program and in direct mode with other C-64 Basic keywords which tested the compatibility of the skeleton program subroutines with the C-64 Basic operating system subroutines.

After the creation and testing of the skeleton emulation program, creating each new Basic keyword was just a matter of creating the new keyword subroutine and adding the newly created subroutine to the skeleton emulation program. The new Basic keyword subroutines were tested by using the new keywords in a Basic program or in direct mode, then verifying that the results of the new Basic keywords were as expected when RUNning the program or executing the new Basic keyword in direct mode.

Sometimes the new Basic keyword added to the emulation program would require a modification of an existing skeleton subroutine or addition of other skeleton subroutines. One example of this type of modification is when the new Basic keyword EXEC subroutine was added to the emulation program. This new Basic keyword required the adding of the MAINA subroutine to the emulation program in addition to the EXEC subroutine. The modification of existing routines requires the testing of the newly added Basic keyword, but also of all other keywords added previously.

Due to the interaction of all new Basic keywords and the original C-64 Basic keywords in the emulation programs complete testing of all possible usage variations of the keywords becomes impossible. Therefore extensive testing but not complete testing of each added keyword was done. The testing of each new Basic keyword consisted of using each keyword in a program and in direct mode. Logically selected keywords were also included in the testing programs to verify that the keywords work in conjunction with each other. All compatibility differences of the emulated commands found during testing that could not be eliminated because of hardware or software constraints are fully described in Appendix A - The TransVersion User Guide. These differences do not seriously affect performance of the emulation software, however, they can cause some debugging problems during execution of the emulated commands within programs which thus, limits the usefulness of the software.

The test procedures for the individual Basic keywords contained tests for errors that might occur because of programming errors, syntax and logical errors, and user interrupt errors, the pressing of the stop key or the run/stop - restore keys. Thus user generated errors were tested for during each testing phase. For example,

during testing of the emulation software in the HGR2 mode and the execution of the HPLOT command an error in the value of the end points or the pressing of the stop key could cause an error written to the Text screen not being displayed. This error was encountered and corrected by adding to the display error routine the applicable code to return the screen display to the TEXT mode before displaying any errors. This error condition was repeated after the addition of the error correction routine to the emulation program and acceptable results were produced.

Although the testing of the TransVersion system software was extensive, the testing was not all encompassing. Thus a fail safe routine was built into the TransVersion system software. If the user inadvertently encounters a software bug (error) or accidentally corrupts the TransVersion system software, either of which could cause the loss of control of the computer, by pressing the run/stop - restore keys the user will usually regain control of the computer. The run/stop - restore keys will cause the CPU of the C-64 to execute a nonmaskable interrupt (NMI). The NMI routine modified with a fail safe routine by the TransVersion system software will perform a checksum on the TransVersion system software. If the checksum is correct then the TransVersion system software is enabled. Otherwise if the checksum is not

correct an error message is printed and a normal run/stop - restore sequence occurs. This fail safe error routine allows the user at any time to determine if the TransVersion system software is intact allowing the user to recover from most errors, either user errors or software errors.

V. CONCLUSIONS AND RECOMMENDATIONS

The design, development, and testing of the TransVersion system has led to several conclusions:

1. The TransVersion system transfers Apple files to the Commodore 64 quickly and accurately. The TransVersion system reduces the time to transfer and convert an Applesoft Basic program to a C-64 program from hours to minutes.

2. Since the TransVersion system allows the user to convert Applesoft Basic files to Commodore Basic files semiautomatically, the resultant Commodore Basic file is more accurate and thus more easily corrected than the resultant Basic program of the usually error-prone manual entry process.

Although the TransVersion system allows for easy transfers and conversion of Apple files to the Commodore 64, it is recommended that future revisions of the TransVersion system be made to correct several of the deficiencies that now occur in the present version of the TransVersion system. The recommendations are listed below.

1. Modify the TransVersion system's Apple emulation software to include the Applesoft commands not currently supported by the TransVersion system's Apple emulation software. These commands are listed in Table XIX in Appendix A - The TransVersion User Guide.

2. Modify the TransVersion system's Apple emulation software to correct any difference in the Applesoft commands and the C-64 emulated Applesoft commands. The commands that are different are listed in Table XXII in Appendix A - The TransVersion User Guide. The differences between the Apple commands and the C-64's emulated Apple commands are described in Appendix A - The TransVersion User Guide.

3. Modify the TransVersion system's emulation software to allow file names with embedded keywords to be used with the emulated Apple DOS disk commands. This problem is described fully in the disk command chapter of Appendix A - The TransVersion User Guide.

4. Modify the TransVersion system's transfer and conversion software to support Apple DOS random access file disk commands.

5. Modify the TransVersion system's transfer software to display the line numbers of all lines commented out for easier identification of those lines that need further modification (editing) by the user.

These recommendations are relatively easy to implement and would allow for more efficient conversions of Applesoft Basic programs to Commodore 64 Basic programs.

LIST OF REFERENCES

1. Espinosa, Christopher, Apple II Reference Manual, Apple Computer, Inc., Calif., 1979, p. 100.

2. Ibid., pp. 23, 24, 79, 100.

3. Ibid., pp. 98, 99, Schematic.

4. Ibid., pp. 79, 98, 99, Schematic.

5. Ibid., pp. 78.

6. Commodore 64 Programmer's Reference Guide, first edition, Commodore Business Machines, Inc., PA, 1984, pp. 359, 360.

7. Ibid., pp. 360,361.

8. Luebbert, Williams F., What's Where in the Apple - An Atlas to the Apple Computer, Micro Ink, Inc., Mass., 1981, pp. $00F2 - $0325.

9. Ibid., pp. $0098 - $00CE.

10. Espinosa, pp. 23, 24.

11. Worth, Don and Lechner, Pieter, Beneath Apple DOS, Quality Software, 1985, pp. 4 - 10.

12. DOS Programmer's Manual, Apple Computer, Inc., Calif., 1982, p. 150.

13. Leemon, Sheldon, Mapping the Commodore 64, Compute! Publications, Inc., NC., 1984, p. 82.

14. Commodore 64 Programmer's Reference Guide, pp. 359-362.

15. Espinosa, p. 88.

16. Commodore 64 Programmer's Reference Guide, pp. 450, Schematic

17. Applesoft Basic Programming Reference Manual, Apple Computer Inc., Calif., 1981, p. 121.

18. Leemon, pp. 89 - 92.

19. Commodore 64 Programmer's Reference Guide, pp. 94 - 95.

20. Leemon, pp. 68.

21. Ibid., pp. 68 - 76.

22. The Second Book of the Commodore 64, Compute! Publications, Inc., NC, 1984, pp. 194 - 198.

23. Leemon, pp. 68 - 76.

24. DOS Programmer's Manual, p. 177.

25. Espinosa, p. 89.

26. The DOS Manual, Apple Computer Inc., Calif., 1981, p. 50.

27. Commodore 1541 Disk Drive User's Guide, Commodore Business Machines Electronics Ltd., England, 1982, pp. 20 - 24.

28. Commodore 64 Programmer's Reference Guide, pp. 94 - 95.

29. Applesoft Basic Programming Reference Manual, Quick Reference Guide.

30. Commodore 64 Programmer's Reference Guide, pp. 12 - 13.

31. Heiserman, David L., Apple IIe Programmer's Reference Guide, Howard W. Sams & Co., Inc., Indiana, 1984, pp. 53 - 54.

32. Commodore 64 Programmer's Reference Guide, pp. 13 - 15.

33. Ibid., pp. 13.

34. Applesoft Basic Programming Reference Manual, pp. 66 - 67.

35. Commodore 64 Programmer's Reference Guide, pp. 54 - 55.

36. Applesoft Basic Programming Reference Manual, pp. 67 - 68.

37. <u>Commodore 64 Programmer's Reference Guide,</u> pp. 22 - 24; 49 - 50.

38. <u>Applesoft Basic Programming Reference Manual,</u> pp. 87 - 88.

39. <u>Commodore 64 Programmer's Reference Guide,</u> pp. 121 - 127.

40. <u>Applesoft Basic Programming Reference Manual,</u> pp. 89 - 90.

41. <u>Commodore 64 Programmer's Reference Guide,</u> pp. 121 - 127.

42. Heiserman, pp. 180 - 181.

43. <u>The DOS Manual,</u> pp. 92, 147 - 164.

44. <u>Ibid.,</u> p. 93.

45. <u>Commodore 64 Programmer's Reference Guide,</u> pp. 35 - 92.

46. <u>Applesoft Basic Programming Reference Manual,</u> pp. 38 - 103.

47. <u>The DOS Manual,</u> pp. 92, 147 - 164.

48. <u>Commodore 1541 Disk Drive User's Guide,</u> pp. 8 - 24, 36 - 44.

BIBLIOGRAPHY

Angerhausen, M., Becker, A., Englisch, L., and Gerits, K., The Anatomy of the Commodore 64, fourth printing, Abacus Software, Inc., MI., 1984.

Applesoft Basic Programming Reference Manual, Apple Computer Inc., Calif., 1981.

Commodore 1541 Disk Drive User's Guide, Commodore Business Machines Electronics Ltd., England, 1982.

Commodore 64 Programmer's Reference Guide, first edition, tenth printing, Commodore Business Machines, Inc., PA, 1984.

Compute's First Book of the Commodore 64, Compute! Publications, Inc., NC., 1983.

Crider, Bill, Basic Program Conversions, Hpbooks, Inc., Arizona, 1984.

DOS Programmer's Manual, Apple Computer, Inc., Calif., 1982.

Englisch, Lothar, The Machine Language Book for the Commodore 64, second printing, Abacus Software, Inc., MI., 1984.

Englisch, Lothar and Szczepanowski, Norbert, The Anatomy of the 1541 Disk Drive, third english printing, Abacus Software, Inc., MI., 1984.

Espinosa, Christopher, Apple II Reference Manual, Apple Computer, Inc., Calif., 1979.

Heiserman, David L., Apple IIe Programmer's Reference Guide, first edition, first printing, Howard W. Sams & Co., Inc., Indiana, 1984.

Immers, Richard, Ph.D, and Neufeld, Gerald G., Ph.D, second printing, Datamost, Inc., Calif., 1985.

Leemon, Sheldon, Mapping the Commodore 64, Compute! Publications, Inc., NC., 1984.

Luebbert, Williams F., What's Where in the Apple - An Atlas to the Apple Computer, Micro Ink, Inc., Mass., 1981.

Mansfield, Richard, Machine Language for Beginners, Compute! Publications, Inc., NC., 1983.

Mansfield, Richard, The Second Book of Machine Language, Compute! Publications, Inc., NC., 1984.

The DOS Manual, Apple Computer Inc., Calif., 1981.

The Second Book of the Commodore 64, Compute! Publications, Inc., NC, 1984.

Worth, Don and Lechner, Pieter, Beneath Apple DOS, seventh printing, Quality Software, 1985.

APPENDIX A*

TRANSVERSION USER GUIDE

* This Appendix contains the TransVersion User Guide, written by the author to assist the user with the TransVersion System software. The user guide contains its own Table of Contents, Figures, Tables, and Appendices. The Table of Contents of the user guide has been changed to reflect the page numbers of this document for the convenience of the reader.

PREFACE

This software/hardware package allows the user to transfer all types of files from an Apple II series computer to a Commodore 64 computer. The simple interface and quick transfer rate can save many man-hours of program typing and editing. In addition to Basic and machine language programs, text files such as those used in word processing programs can be transferred, thus allowing the user to transfer an entire data base from the Apple to the Commodore C-64. Below are the principal features of this package.

- Simple interface hookup
- Quick transfer rate
- Graphics commands available
- Sequential file commands allowed
- Apple character set option available
- Many Apple commands emulated
- "REM out" of non-implemented program lines
- Easy identification of non-implemented commands
- Quick reference memory maps
- Quick reference Basic conversion Tables

The software allows all common and emulated Basic command lines to transfer quickly, correcting any syntax differences. POKE, PEEK, and unavailable commands are commented out (i.e. REM) and the unimplemented command is identified for editing purposes. The conversion tables in the Appendix allow the programmer ready reference to the Applesoft and Commodore 64 Basic command words.

Many Applesoft commands not available on Commodore 64 can be transferred with a technique called emulation programming. Emulation programming, which uses a background machine language program, will allow new Basic command words to be implemented on the Commodore 64. This saves many man-hours in writing Basic and machine language routines to implement unavailable Applesoft commands on the Commodore 64.

TABLE OF CONTENTS

Page

I. DESCRIPTION OF TRANSFER PACKAGE

Upon receiving your transfer package, verify that each item listed in Table I is included.

TABLE I

PARTS LISTS

1- Apple Transfer Diskette
1- Commodore 64 Transfer diskette
1- Transfer interface cable
1- Transfer Package User Guide (this book)

The Apple Transfer Diskette contains:

FILENAME	TYPE	USAGE
MLBASICTRANSFER	binary	Basic program file transfer driver
MLBINTRANSFER	binary	Binary file transfer driver
MLTEXTTRANSFER	binary	Text file transfer driver
HELLO	Basic	Apple auto boot file menu program

The Commodore 64 Transfer Diskette contains:

FILENAME	TYPE	USAGE
TRANSFERA	binary	Transfer program receiver (section a)
TRANSFERB	binary	Transfer program receiver (section b)
EMULATEA	binary	Implements new commands (section a)
EMULATEB	binary	Implements new commands (section b)
CHARACTERSET	binary	Installs Apple character set
MENU	Basic	Loads system software/main boot program
BOOT ALLA	Basic	Loads characterset software options

Note: It is recommended that you make backup copies of both diskettes before attempting program transfer. Store the source diskettes in a safe place. Use copies of the source diskettes as your working diskettes.

II. SETTING UP FOR TRANSFER

The procedure for setting up for transfer has been made as simple as possible. Be sure to follow the steps shown below carefully or damage to computers could occur.

A. Connecting the Interface Hardware

A.) Turn off both computers. Remove all game cartridges and disk cartridges including "Fast Load" cartridges. Disk drives and printers need not be removed.

B.) At the C-64 User Port verify that pin 1 of the computer port and pin 1 of theinterface cable connect properly. Note that the cable leads will be pointing away from the center of the computer when looking toward the back of the computer from the keyboard (i.e. toward left). This is to insure that the plug is not put in backwards.

C.) At the Apple II computer, remove the cover on the Apple II, thread the interface cable through an available opening in back of the Apple computer and connect the Apple end of the interface cable to the Apple paddle port. The Apple paddle port is a 16 pin dual-in-line package socket located just to the right of the input/output (I/O) slot 6 near the back of the Apple II computer.[25] Be sure pin 1 on the interface cable is aligned with pin 1 on the Apple paddle port socket. Note that the interface cable

will point toward the back of the Apple computer when it is properly installed.

D.) Recheck all connections.

E.) Replace the top cover on the Apple computer.

F.) Turn on the Commodore 64; the normal power up logo should appear. If it does not, turn the computer off immediately, and recheck both computer interface connections.

G.) Insert a copy of the Apple Transfer diskette into the Apple disk drive. Switch on the Apple computer. The Apple should boot up properly. It will automatically install the transfer program and will guide you through the transfer process.

B. <u>Making A Transfer</u>

A.) Follow the directions as shown by the Apple program until told to activate the software on the Commodore 64 computer.

B.) On the Commodore 64, insert a copy of the C-64 Transfer diskette. Load and Run the basic program named MENU (LOAD "*",8:RUN or LOAD "MENU",8:RUN). Answer all the questions. The simplest program transfer will occur if all questions are answered with defaults.

C.) The C-64 program will load the machine language transfer receiver program. The program will also set up software options requested by the operator. These options are described later in this book.

D.) After all questions are answered on the Commodore 64, the following message will be displayed: TRANSFER PROGRAM WAITING ON APPLE. CONTINUE WITH APPLE PROGRAM.

E.) On the Apple computer, the operator is requested to enter the name of the file, and the type of file. Then the user inserts the diskette containing the file to be transferred. The Apple program will transfer the requested file then return control to the main menu program.

F.) The C-64 program will save the transferred file to the diskette and return to its main menu for another program transfer.

III. DESCRIPTION OF SOFTWARE OPTIONS

A. Transfer Mode Options

The following sections will describe in detail the software options and processes during the transfer of a program/data file from the Apple II to the Commodore 64 (C-64).

1. Text File Transfers

In some cases, a data base is stored on an Apple formated diskette, and it maybe desirable to transfer the data base file to the Commodore 64. This capability is provided in the accompanying software. Word processing files and assembler source code files are two examples of data text files that may be usefully transferred.

Selecting the Text File Transfer option allows the operator to transfer Applesoft sequential and random access files. The Apple program will request from the operator the type of file to be transferred (random access or sequential). If the file type selected is random access, the Apple program will request the record length. The Apple will then transfer the text file sector by sector until the end of the file is reached. The Apple program will display the contents of the text file as it is being transferred to the C-64. The C-64 program will

receive the data and store the file on disk a sector at a time during transfer. The program will return to the MENU program after the complete file is transferred. This procedure allows the maximum length text files to be transferred to the C-64.

2. <u>Binary File Transfers</u>

Machine language programs and screen images (graphics) are frequently stored on the Apple diskette as binary files. Although changes to these files are usually necessary to get them to work properly on the Commodore 64, transferring the files is frequently more desirable than recreating them from scratch. Therefore a binary file transfer option is provided for your use.

The Binary File Transfer option will allow binary files to be transferred from the Apple II to the Commodore 64. The Apple program retrieves the starting address and length from the selected binary file. The binary file is then BLOADed to location 10000 decimal; then the file is transferred to the C-64. If the binary program is longer than 28400 bytes the program will overwrite Apple'S DOS and the computer will crash. There is no check to see if the file is too long. Therefore care must be taken to be sure the binary file is not too large.

The C-64 program will retrieve the file start address and the file length from the Apple computer. A

check is made to see if the file can be stored at the original starting address. Start addresses between $400 and $94FF are acceptable. If the start address falls outside this range, the program relocates the start address to start of Basic program memory area ($801 hex). The program is stored in memory until the complete file is transferred to the C-64 memory. The C-64 program then SAVEs the binary program to the diskette and returns to the MENU program. The program doesn't display any error condition from the disk drive that might occur.

3. <u>Basic Program File Transfers</u>

a. <u>Description and Operation</u> Selecting this option allows the operator to transfer Applesoft Basic program files to the C-64 computer. This option is the most complicated type of transfer, because it allows for versatility in the type of transfer that is best suited to the operator's preferences or applications program necessities.

The simplest Basic transfer occurs when no additional options are selected. This section will describe in detail the simplest Basic transfer and the special features of the transfer software. The following sections will describe the additional options that can be selected and their effects on the transfer process.

The C-64 transfer receiver program handles the conversion of the Applesoft Basic commands to C-64 Basic commands. The basic operation of the receiver program is listed in Table II.

TABLE II

RECEIVER PROGRAM OPERATIONAL DESCRIPTION

Before Transfer

1. Store option lines in memory if necessary. See Emulation option section.

During Transfer

2. Receives data bytes from Apple computer.
3. Stores all line numbers, pointers, and characters directly in memory.
4. Converts all Apple commands to C-64 commands then stores the equivalent C-64 commands in memory.
5. REM all lines containing incompatible commands and identifies the incompatible commands.

After Transfer

6. Converts all recognizable Apple disk commands to the proper C-64 disk commands.
7. Divides and splits all the program lines longer than 80 characters into lines shorter than 80 characters, if possible.
8. SAVEs the transferred program to the diskette.
9. LOADs and RUNs the MENU program from the diskette.

The transfer software converts the Applesoft commands in Table IV to C-64 commands during transfer. Generally, most of these commands will not cause problems during RUNning of the transferred program, however, some situations will occur which will cause the transferred program not to work properly. The differences in the C-64 and Apple II commands are further detailed in the Emulation Mode chapters and Appendix A.

The Applesoft commands listed in Table III are unavailble on the C-64 and will be commented out (REM) during transfer. These unimplemented commands will also be identified with the underline character being inserted immediately before and after the command.

TABLE III

APPLESOFT COMMANDS COMMENT OUT (REM)

AT	POKE
CALL	PR#
COLOR=	RECALL
DEL	RESUME
DRAW	ROT=
GR	SCALE=
HLIN	SCRN(
IN#	SHLOAD
ONERR	STORE
PEEK	VLIN
PLOT	XDRAW

TABLE IV

APPLESOFT COMMANDS DIRECTLY TRANSFERRED

ABS	INPUT	POS
AND	INT	PRINT
ASC	LEFT$	READ
CHR$	LEN	REM
CLEAR (CLR)	LET	RESTORE
COS	LIST	RETURN
DATA	LOG	RIGHT$
DEF	MID$	RND
DIM	NEW	SGN
END	NOT (0=	SIN
EXP	TAN	SPC
FOR	USR	SQR
FRE	VAL	STEP
GOSUB	WAIT	STOP
GOTO	ON	STR$
IF	OR	TAB

DISK COMMANDS

CLOSE	RUN (C-64 LOAD)	NOMON C
DELETE	SAVE	
READ	WRITE	
OPEN	MON C	

WITH EMULATION OPTION SELECTED

BRUN	HGR2	NOTRACE
BLOAD	HIMEM:	PDL
BSAVE	HOME	POP
EXEC	HPLOT	SPEED=
FLASH	HTAB	TEXT
GET (GIT)	INVERSE	TRACE
HCOLOR=	LOMEM:	VTAB
HGR	NORMAL	CATALOG

b. <u>Disk Commands Translation</u> After the transfer is complete, a second pass through the C-64 program in memory is performed to convert all the recognizable Apple

disk commands to the equivalent C-64 disk commands. The three recognizable disk command identifiers are D$,CHR$(4), and the embedded control-d. The three recognizable disk command types are shown in Table V.

TABLE V

RECOGNIZABLE APPLESOFT DISK COMMANDS

```
1.) PRINT D$,"DISK COMMAND" : REM  D$ IS CONTROL-D
2.) PRINT D$;"DISK COMMAND"+"DISK OPTIONS"
3.) PRINT D$"DISK COMMAND";B$:REM B$="DISK OPTIONS"
4.) PRINT CHR$(4),"DISK COMMAND"
5.) PRINT CHR$(4)"DISK COMMAND" B$:REM B$="FILENAME
    AND DISK OPTIONS"
6.) PRINT CHR$(4);"DISK COMMAND",B$:
    REM B$="DISK OPTIONS"
7.) PRINT " DISK COMMAND" : REM EMBEDDED CONTROL-D
8.) PRINT " DISK COMMAND"+ B$:REM EMBEDDED CONTROL-D
9.) PRINT D$ "DISK COMMAND" + "FILENAME" + B$ :
    REM B$="DISK OPTIONS"
```

All valid syntax variations (i.e. semicolon, comma, plus and none) of the formats above are recognized. All recognizable disk commands have the actual disk command embedded into the quote string (i.e. "READ","OPEN FILENAME"). The filename and disk options may be in the form of a Basic variable or in the quote string itself. Other formats will not be recognized as valid Apple disk commands. Two such unrecognized formats are shown below.

1.) PRINT A$,"DISK COMMAND" : REM A$=CONTROL-D

2.) PRINT D$+B$:REM B$="DISK COMMAND"

Note: D$ is the only string variable recognized as a control-d. String variables are not recognized as disk command strings.

The Apple DOS 3.3 disk commands that are converted to C-64 Basic syntax are listed in Table VI.

TABLE VI
APPLE DISK COMMANDS CONVERTED TO C-64 SYNTAX

OPEN	SAVE
DELETE	RUN (C-64 LOAD)
WRITE	MON C
CLOSE	NOMON C
READ	
EMULATION MODE ONLY	
BLOAD	BSAVE
BRUN	EXEC
CATALOG	

The Apple DOS 3.3 disk commands that are not supported and commented out (REM) are listed in Table VII.

TABLE VII

UNSUPPORTED APPLE DOS 3.3 DISK COMMANDS

POSITION	MAXFILE	INT
APPEND	LOAD	FP
VERIFY	MON I,O	IN#
NOMON I,O	PR#	

For fully detailed descriptions of each of these commands see Appendix B. The Emulation Mode disk commands are also detailed separately in the Emulation Mode section.

The processing of the Apple disk commands will be affected by the contents of the program. The Applesoft disk commands MON and NOMON will determine how the conversion will be accomplished. See Table IX and Table X for the effects of the Apple disk commands MON C and NOMON C on the disk command processing. The conversion process defaults to NOMON C processing if the Apple disk command MON C is not present (MON I,O AND NOMON I,O ARE IGNORED). Table VIII shows the original Applesoft program. See Appendix B for more details of Apple disk commands.

TABLE VIII

ORIGINAL APPLE PROGRAM FOR TABLES IX AND X

```
10  REM DISK COMMAND EXAMPLE
20 REM MON PROCESSING
25 PRINT
30 D$=CHR$(4)
40 PRINT CHR$(4);"MONICO": REM OR NOMONICO
45 REM DISK COMMANDS IMPLEMENTED BY  EMULATION OPTION
50 PRINT " CATALOG": REM EMBEDDED CONTROL-D
60 PRINT D$;"BSAVE SCREEN,A1024,L1"
70 PRINT D$;"BRUN SCREEN,A1024"
80 PRINT D$;"BLOAD SCREEN,A$401"
90 PRINT D$;"EXEC  SCREEN"
95 REM DISK COMMANDS ALWAYS CONVERTED
100 PRINT D$;"OPEN TEXT FILE"
103 PRINT D$;"DELETE TEXT FILE"
106 PRINT D$;"OPEN TEXT FILE"
110 PRINT D$;"WRITE TEXT FILE"
112 PRINT "STORED IN FILE"
114 INPUT A$
120 PRINT D$;"CLOSE TEXT FILE"
122 PRINT D$;"OPEN TEXT FILE"
124 PRINT D$;"READ TEXT FILE"
126 INPUT A$
128 PRINT A$
129 PRINT D$;"CLOSE TEXT FILE"
130 PRINT "SAVE PROGRAM":REM EMBEDDED CONTROL-D
140 PRINT CHR$(4);"RUN PROGRAM"
150 REM DISK COMMANDS NOT CONVERTED
160 PRINT D$;"POSITION TEXT FILE,R1"
170 PRINT D$;"APPEND TEXT FILE"
180 PRINT D$;"VERIFY PROGRAM:"
190 PRINT D$;"LOAD PROGRAM"
200 PRINT D$;"MAXFILE 3"
210 REM END OF PROGRAM
220 END
```

TABLE IX

TRANSFERRED C-64 PROGRAM WITH

MON IN EFFECT (NO EMULATION)

```
10  REM DISK COMMAND EXAMPLE
20  REM MON PROCESSING
25  PRINT
30  D$=CHR$(4)
40 REMPRINT CHR$(4);"MONICO": REM OR NOMONICO
45  REM DISK COMMANDS IMPLEMENTED BY  EMULATION OPTION
50 REMPRINT " CATALOG": REM EMBEDDED CONTROL-D
60 REMPRINT D$;"BSAVE SCREEN,A1024,L1"
70 REMPRINT D$;"BRUN SCREEN,A1024"
80 REMPRINT D$;"BLOAD SCREEN,A$401"
90 REMPRINT D$;"EXEC  SCREEN"
95  REM DISK COMMANDS ALWAYS CONVERTED
100  PRINT D$;"OPEN TEXT FILE"
103  PRINT D$;"DELETE TEXT FILE":OPEN15,8,15,
     "S0:TEXT FILE":CLOSE 15
106  PRINT D$;"OPEN TEXT FILE"
110  PRINT D$;"WRITE TEXT FILE":OPEN 14,8,14,
     "0:TEXT FILE,S,W"
112  PRINT#14,"STORED IN FILE"
114  INPUT A$
120  PRINT D$;"CLOSE TEXT FILE":CLOSE 14
122  PRINT D$;"OPEN TEXT FILE"
124  PRINT D$;"READ TEXT FILE":OPEN 14,8,14,
     "0:TEXT FILE,S,R"
126  INPUT#14,A$
128  PRINT A$
129  PRINT D$;"CLOSE TEXT FILE":CLOSE 14
130  PRINT "SAVE PROGRAM":REM EMBEDDED CONTROL-D:
     SAVE "PROGRAM",8
140  PRINT CHR$(4);"RUN PROGRAM":LOAD "PROGRAM",8
150  REM DISK COMMANDS NOT CONVERTED
160 REMPRINT D$;"POSITION TEXT FILE,R1"
170 REMPRINT D$;"APPEND TEXT FILE"
180 REMPRINT D$;"VERIFY PROGRAM:"
190 REMPRINT D$;"LOAD PROGRAM"
200 REMPRINT D$;"MAXFILE 3"
210  REM END OF PROGRAM
220  END
```

TABLE X

TRANSFERRED C-64 PROGRAM WITH NOMON

IN EFFECT (NO EMULATION)

```
10  REM DISK COMMAND EXAMPLE
20  REM NOMON PROCESSING
25  PRINT
30  D$=CHR$(4)
40 REMPRINT CHR$(4);"MONICO": REM OR NOMONICO
45  REM DISK COMMANDS IMPLEMENTED BY  EMULATION OPTION
50 REMPRINT " CATALOG": REM EMBEDDED CONTROL-D
60 REMPRINT D$;"BSAVE SCREEN,A1024,L1"
70 REMPRINT D$;"BRUN SCREEN,A1024"
80 REMPRINT D$;"BLOAD SCREEN,A$401"
90 REMPRINT D$;"EXEC  SCREEN"
95  REM DISK COMMANDS ALWAYS CONVERTED
100 REMPRINT D$;"OPEN TEXT FILE"
103  OPEN15,8,15,"S0:TEXT FILE":CLOSE 15
106 REMPRINT D$;"OPEN TEXT FILE"
110  OPEN 14,8,14,"0:TEXT FILE,S,W"
112  PRINT#14,"STORED IN FILE"
114  INPUT A$
120  CLOSE 14
122 REMPRINT D$;"OPEN TEXT FILE"
124  OPEN 14,8,14,"0:TEXT FILE,S,R"
126  INPUT#14,A$
128  PRINT A$
129  CLOSE 14
130  SAVE "PROGRAM",8
140  LOAD "PROGRAM",8
150  REM DISK COMMANDS NOT CONVERTED
160 REMPRINT D$;"POSITION TEXT FILE,R1"
170 REMPRINT D$;"APPEND TEXT FILE"
180 REMPRINT D$;"VERIFY PROGRAM:"
190 REMPRINT D$;"LOAD PROGRAM"
200 REMPRINT D$;"MAXFILE 3"
210  REM END OF PROGRAM
220  END
```

These previous examples illustrate the effect of MON C and NOMON C. If the transfer software encounters a

MON C command the transfer software doesn't comment out (REM) the original Applesoft disk command lines following the MON command and thus the disk command is printed to the screen when the program is RUN. The equivalent C-64 disk command will immediately follow the original Applesoft command line. If the transfer software encounters a NOMON C command the following original Applesoft disk command line is replaced with the C-64 disk command and thus the disk command doesn't get printed to the screen during RUNning of the program. This technique results in a similar effect that NOMON C and MON C have on the Applesoft program during RUNning of the program.

The disk command processing is also affected by the Emulation option. The Emulation option adds several disk commands not normally available using C-64 Basic; thus the Emulation option will affect the overall disk command conversion. See Table XI and Table XII for the effects the Emulation option has on the disk command processing. The Emulation option affects only the conversion of the new C-64 disk command implemented by the Emulation option. See the Emulation Mode chapter for further details on Emulation Mode disk commands.

TABLE XI

EMULATION OPTION EFFECTS ON THE DISK COMMAND

PROCESSING WITH MON IN EFFECT

```
10  REM DISK COMMAND EXAMPLE
20  REM MON PROCESSING
25  PRINT
30  D$=CHR$(4)
40 REMPRINT CHR$(4);"MONICO": REM OR NOMONICO
45  REM DISK COMMANDS IMPLEMENTED BY  EMULATION OPTION
50 PRINT " CATALOG":CATALOG
60  PRINT D$;"BSAVE SCREEN,A1024,L1":BSAVE SCREEN,A1024,L1
70  PRINT D$;"BRUN SCREEN,A1024":BRUN SCREEN,A1024
80  PRINT D$;"BLOAD SCREEN,A$401":BLOAD SCREEN,A$401
90  PRINT D$;"EXEC  SCREEN":EXEC SCREEN
95  REM DISK COMMANDS ALWAYS CONVERTED
100  PRINT D$;"OPEN TEXT FILE"
103  PRINT D$;"DELETE TEXT FILE":OPEN15,8,15,"S0:TEXT
     FILE":CLOSE 15
106  PRINT D$;"OPEN TEXT FILE"
110  PRINT D$;"WRITE TEXT FILE":OPEN 14,8,14,
     "0:TEXT FILE,S,W"
112  PRINT#14,"STORED IN FILE"
114  INPUT A$
120  PRINT D$;"CLOSE TEXT FILE":CLOSE 14
122  PRINT D$;"OPEN TEXT FILE"
124  PRINT D$;"READ TEXT FILE":OPEN 14,8,14,
     "0:TEXT FILE,S,R"
126  INPUT#14,A$
128  PRINT A$
129  PRINT D$;"CLOSE TEXT FILE":CLOSE 14
130  PRINT "SAVE PROGRAM":REM EMBEDDED CONTROL-D
     :SAVE "PROGRAM",8
140  PRINT CHR$(4);"RUN PROGRAM":LOAD "PROGRAM",8
150  REM DISK COMMANDS NOT CONVERTED
160 REMPRINT D$;"POSITION TEXT FILE,R1"
170 REMPRINT D$;"APPEND TEXT FILE"
180 REMPRINT D$;"VERIFY PROGRAM:"
190 REMPRINT D$;"LOAD PROGRAM"
200 REMPRINT D$;"MAXFILE 3"
210  REM END OF PROGRAM
220  END
```

TABLE XII

EMULATION OPTION EFFECTS ON THE DISK COMMAND

PROCESSING WITH NOMON IN EFFECT

```
10  REM DISK COMMAND EXAMPLE
20  REM NOMON PROCESSING
25  PRINT
30  D$=CHR$(4)
40 REMPRINT CHR$(4);"NOMONICO": REM OR NOMONICO
45  REM DISK COMMANDS IMPLEMENTED BY  EMULATION OPTION
50  CATALOG
60 BSAVE SCREEN,A1024,L1
70 BRUN SCREEN,A1024
80 BLOAD SCREEN,A$401
90 EXEC  SCREEN
95  REM DISK COMMANDS ALWAYS CONVERTED
100 REMPRINT D$;"OPEN TEXT FILE"
103  OPEN15,8,15,"S0:TEXT FILE":CLOSE 15
106 REMPRINT D$;"OPEN TEXT FILE"
110  OPEN 14,8,14,"0:TEXT FILE,S,W"
112  PRINT#14,"STORED IN FILE"
114  INPUT A$
120  CLOSE 14
122 REMPRINT D$;"OPEN TEXT FILE"
124  OPEN 14,8,14,"0:TEXT FILE,S,R"
126  INPUT#14,A$
128  PRINT A$
129  CLOSE 14
130  SAVE "PROGRAM",8
140  LOAD "PROGRAM",8
150  REM DISK COMMANDS NOT CONVERTED
160 REMPRINT D$;"POSITION TEXT FILE,R1"
170 REMPRINT D$;"APPEND TEXT FILE"
180 REMPRINT D$;"VERIFY PROGRAM:"
190 REMPRINT D$;"LOAD PROGRAM"
200 REMPRINT D$;"MAXFILE 3"
210  REM END OF PROGRAM
220  END
```

The above examples illustrate the effects of disk command processing with Emulation option selected. The

Emulation option allows the implementation of five additional Applesoft disk commands (ref. Table VI) and will not comment out (REM) the Applesoft disk command lines containing the newly implemented commands. MON C and NOMON C processing convert the newly implemented command lines under the same criteria as previously described.

The Apple disk commands for input/output (I/O) operations work differently between the Apple II and C-64 computers. When the Apple Dos 3.3 READ command is executed by a RUNning program all subsequent input commands (INPUT or GET) encountered take their response from the appropriate disk file and data is not retrieved from the keyboard. When the Apple Dos 3.3 WRITE command is executed by a RUNning program all subsequent PRINT commands write data to the disk file and not the screen.[26] The C-64 will use the commands INPUT#, GET#, and PRINT# to read/write to the file on disk that has been previously OPENed for reading or writing.[27] Due to these differences, the transfer software will convert all GET and INPUT commands following an Apple Dos 3.3 READ command to GET# and INPUT# until an Apple CLOSE command is encountered. Likewise the transfer software will convert all PRINT commands following an Apple WRITE command to PRINT# commands until an Apple CLOSE command is

encountered.

The C-64 disk command OPEN and the Apple Dos 3.3 OPEN disk command operate differently. The C-64 disk command OPEN requires the file to be OPENed for reading or writing. This information is not available to the transfer software until an Apple Dos 3.3 READ or WRITE command is encountered. Thus, the original Apple OPEN program line is commented out (REM). When the transfer software encounters the Apple Dos 3.3 READ or WRITE command, the C-64 OPEN command will be generated with the proper input/output (I/O) direction. See Table VIII and Table XII lines 106 and 110. This procedure will cause improper Running of the transferred program on the C-64 if the Apple APPEND command is used in the Apple program. The C-64 transferred program will incorrectly start writing at the beginning of the specified file instead of the end of the file. This can be easily corrected by the user before attempting to RUN the transferred C-64 program containing the Apple APPEND command. See the APPEND command in Appendix B.

Due to the above procedures, and because the Apple file structure uses the filename to identify what file is to be active at any given time, and the C-64 uses numbers to identify the active files, the transfer software makes

several assumptions about the contents of the Apple program. These assumptions are listed in Table XIII.

TABLE XIII

APPLE PROGRAM ASSUMPTIONS

1.) The program has only one file active at any one time.

2.) All file commands are either binary, Basic or sequential

3.) All sequential file commands are assigned the C-64 logical file number of 14.

4.) After a file is OPEN for a Read/Write all I/O commands (PRINT,INPUT and,GET) are converted to C-64 I/O commands (PRINT#, INPUT#, and GET#) until the file is CLOSEd.

These assumptions will be valid in almost all cases. In the situations where more than one file is active at the same time the assumptions used above will allow easy manual editing of the incorrect translation of the disk commands.

Note: All Apple Random Access file commands will be converted to C-64 sequential file commands; thus they will not RUN properly on the C-64.

c. <u>Splitting Long Program Lines</u> After the disk command processing has been accomplished a last pass

through the C-64 program in memory is done to handle all the program lines longer than 80 characters. The C-64 will RUN program lines with a length up to a maximum of 256 characters. However the C-64 will not permit the operator to edit lines longer than 80 characters on the screen.[28] Therefore for ease of editing the transfer software checks each line in the program for lengths greater than 80 characters. If the line has more than 80 characters, and the next line number is greater than the present line number plus one, and the line is easily split, then the transfer software will split the long line into two shorter lines. The transfer software will continue this process until all the lines in the program have been split and are shorter than 80 characters. The software determines that a line is easily split if the line contains colons and the colons do not follow an IF statement. If a line is a REMark statement the line is not split. Table XIV and Table XV illustrates this process.

TABLE XIV

SPLITTING LONG PROGRAM LINES

ORIGINAL APPLESOFT PROGRAM

```
10 DATA January, February, March ,April, May, June,
   July, August, September, November, December, Monday,
   Tuesday, Wednesday, Thursday, Friday

20 DATA Monday,Tuesday,Wednesday,Thursday, Friday:Rem
   this statement will split into two lines because it
   is longer than 80 characters and has a colon
   separator.

30 REM This program calculates the day of the week
   given the month and year and day of the month.
   Remember this program line will not be split because
   there is no colon separating the line.

40 TEXT:HOME:HGR:HPLOT 0,0 TO A,B: HPLOT 0,0 TO 100,100
   : HPLOT 0,0 TO 100,100: REM This line will not be
   split because the next line number is to close to
   the present line number

41 REM Line number to close to previous line number

50 IF A=0 AND B=0 AND C=0 AND D=0 THEN PRINT"This is a
   long line with a if statement ":PRINT " and will not
   split"

60 TEXT:HOME:HGR:HPLOT 0,0 TO A,B: HPLOT 0,0 TO 100,100
   :IF A=0 THEN PRINT "This line will split because the
   colons are before the IF statement."
```

TABLE XV

TRANSFERRED C-64 PROGRAM

```
10 DATA January, February, March ,April, May, June,
   July, August, September, November, December, Monday,
   Tuesday, Wednesday, Thursday, Friday

20 DATA Monday,Tuesday,Wednesday,Thursday, Friday

21 REM this statement will split into two lines because
   it is longer than 80 characters and has a colon
   separator.

30 REM This program calculates the day of the week
   given the month and year and day of the month.
   Remember this program line will not be split because
   there is no colon separating the line.

40 TEXT:HOME:HGR:HPLOT 0,0 TO A,B: HPLOT 0,0 TO 100,100
   : HPLOT 0,0 TO 100,100: REM This line will not be
   split because the next line number is to close to
   the present line number

41 REM Line number to close to previous line number

50 IF A=0 AND B=0 AND C=0 AND D=0 THEN PRINT"This is a
   long line with a IF statement ":PRINT " and will not
   split because of the IF command."

60 TEXT:HOME:HGR:HPLOT 0,0 TO A,B: HPLOT 0,0 TO 100,100

61 IF A=0 THEN PRINT "This line will split because the
   colons are before the IF statement"
```

d. <u>Apple Program Preparatory Procedures</u> From the above explanations of the conversion process there can be derived several preparatory procedures that will allow maximum conversion of Apple programs. These preparatory procedures are listed in Table XVI.

TABLE XVI

APPLE PROGRAM PREPARATORY PROCEDURES

1.) Have the beginning line number of the Apple program start with the minimum value of six. This condition will allow the conversion software room to add the Emulation option lines.

2.) If the Apple program has many long lines, then let the program have an increment between line numbers of at least ten. This condition will allow the transfer software to split the long lines into shorter lines.

3.) Change all the Apple disk commands in the program to the three recognizable disk command formats (i.e. CHR$(4), D$, or embedded control-d). This condition will allow the transfer software to recognize all Apple disk commands.

4.) Change all Apple disk commands in the program so that each disk command is in it's original form (i.e. READ,WRITE) and not in a string variable (i.e. COMMAND$) form.

5.) If possible be sure only one file is active at any time in the Apple program. Be sure no Apple disk commands are random access file commands.

e. Traps and Pitfalls The above procedures will minimize final manual editing of the transferred program. These procedures minimize error that occur during translation/conversion of programs to C-64 Basic syntax. Other errors can occur during program execution due to the Basic operating system differences between the Apple and C-64 computers. This section will detail those differences and develop procedures to help deal with those

differences. The Applesoft Basic commands that can cause problems during running of the Basic program are listed in Table XVII.

TABLE XVII

DISSIMILAR OPERATING COMMANDS BETWEEN APPLESOFT AND C-64

I/O COMMANDS

INPUT	SAVE
LOAD	POKE
PEEK	

LOGIC COMMANDS

NOT	AND
OR	

The main difference between Applesoft Basic and C-64 Basic is the evaluation of logical expressions and the resulting values obtained from a given logical expression. The Apple computer gives values of 0 (false) or 1 (true).[29] The C-64 computer gives values of 0 (false) or -1 (true).[30] This difference will not cause problems on the C-64 unless the Applesoft statement uses the value (0 or 1) of the result as the condition to do an operation verses using the trueness/falseness condition as the condition to do the operation. An example is shown

below. The Applesoft commands to display the current available RAM as a positive value are:

```
10 X=FRE(0) + (FRE(0)<0)*65536
20 PRINT X
```

The equivalent C-64 commands to display the current available RAM as a positive value are:

```
10 X=FRE(0) - (FRE(0)<0)*65536
20 PRINT X
```

The transfer software will transfer the Applesoft command directly to C-64 with no changes. The error will only occur during RUNning of the transferred program. It will be up to the programmer to manually edit the line to the correct version as shown in the C-64 example above before RUNning the transferred program on the C-64. The programmer can eliminate these errors by using only the true/false condition of the logical expression rather than the value of the expression.

Other problems occur with the boolean operators NOT, AND and OR. The implementation of these commands is based on different logical uses of these commands. The Applesoft version of AND and OR results in only two values 0 or 1 (i.e. true/false).[31] The C-64 versions will result in multiple values depending on the operand and expressions.[32] Therefore the Applesoft command line must

use the condition (trueness/falseness) of the AND/OR operation and <u>not</u> the value of the AND/OR operation, so that, the transferred command lines will operate similarly on the C-64. The examples shown below are a few Applesoft command lines that will operate similarly on the C-64.

```
10 IF A=3 AND B=7 THEN PRINT A+B
20 IF A=2 OR B=2 THEN PRINT A+B
30 IF A AND B=7 THEN PRINT A+B
```

Note that in each example the condition of the expression is used to do the branch and not the value of the expression. The examples below will work differently on the Apple and C-64.

```
10  PRINT A AND B
20 IF (A OR B) =1 THEN PRINT "DONE"
```

Note that the value of the AND/OR expression is used in each case.

It should be noted that the C-64 operand with AND, OR, and NOT must be in limits of -32768 to +32767, otherwise an error message results.[33] The Apple II has no such restrictions.

The transfer software converts all Applesoft NOT commands to the C-64 expression '0='. The reason for this conversion is shown below.

The Applesoft command NOT reverses the logic of the expression and results in TRUE (1) only when the expression is zero and a false (0) when the expression is non-zero. Thus the expression NOT A will give a value of 0 (false) for all non-zero values of A and a value of 1 (true) when A is zero. The C-64 command NOT produces the two's complement of the result. The C-64 expression A=NOT X is equivalent to A=(-(x+1)). Although the C-64 NOT command is very different from the Applesoft NOT command, the C-64 expression 0= is very similar to the Applesoft NOT command. The C-64 expression 0=A gives a value of 0 (false) for all non-zero values of A and a value of -1 (true) when A is zero. The C-64 expression 0= is identical to the Applesoft NOT command for all non-zero operands. The condition (trueness/falseness) is identical for all values of the operand. Thus the only time the original Applesoft NOT statement will not operate similarly is when the statement uses the value of the logical expression and the logical expression is true (i.e. 1=Apple / -1=C-64). This error can be easily corrected by the programmer as described previously.

Other problems occur in the Applesoft I/O commands

INPUT and GET. The Applesoft INPUT command will set a string operand to a null if a return key alone is typed. The C-64 INPUT command will leave the string operand at it's current value if a return key alone is typed. The Applesoft INPUT command with a numeric operand will not allow a non-numeric value (including return key alone) and will request a reentry.[34] The C-64 INPUT command with a numeric operand will allow a return key alone and the numeric operand will remain at it's present value. If a non-numeric key is entered a reentry is requested and the numeric operand is zeroed.[35]

The Applesoft GET command will wait for a single key stroke for each operand listed. If the operand is numeric it waits for a numeric key and if a non-numeric key is entered a reentry request will be made. If the operand is a string the return key alone will return a null character.[36] The C-64 GET command does not wait for a key stroke and will return a zero if the operand is numeric and a null if the operand is a string and no keystroke is present. If the operand is numeric any non-numeric key entry will cause a syntax error.[37] See Emulation Mode chapters for further details on the GET (GIT) command.

The Applesoft POKE and PEEK commands operate

identically to the C-64 commands, nevertheless, the address locations usually do not have the same functions due to the differences in the hardware.

4. Emulation Option

a. Description and Operation The Emulation option will allow the programmer to add new command keywords (new program instructions) to the Commodore 64 Basic keyword list. The keywords added are listed in Table XVIII. These new command keywords will simplify the editing of the transferred programs. The Emulation option will create commands ordinarily available only by machine language programming. The Emulation option will create program lines to be inserted into the beginning of the transferred program. These newly created program lines will LOAD and RUN the emulation software. This will allow the operator to RUN transferred programs stand alone with no need to selectively LOAD emulation software independently. However the emulation program must reside on the diskette in the disk drive when the application program is RUN.

The main advantages of the Emulation option is the implementation of Apple commands that are not available in the C-64 Basic operating system. This implementation of new Basic commands is done by a technique called emulation

TABLE XVIII

APPLE COMMANDS IMPLEMENTED BY EMULATION OPTION

BRUN	KILL
BLOAD	HTAB
BSAVE	INVERSE
EXEC	LOMEM:
FLASH	NORMAL
GET (GIT)	NOTRACE
HCOLOR=	PDL
HGR	POP
HGR2	SPEED=
HIMEM:	TEXT
HOME	TRACE
HPLOT	VTAB
CATALOG	

programming. Emulation programming allows a machine language program to run in the "background" of a Basic program to enhance the capabilities of the host's computer Basic operating system. A familiar example of this is the C-64 DOS Wedge support program.

The machine language routines (emulation programs) used to add the new Basic commands must be in memory and run before any of the new Basic commands will work. Specifically, any program containing these new commands will not RUN unless the emulation programs are residing in memory. Thus, to make the programs transferred from the Apple RUN stand alone, the transfer software adds new program lines at the beginning of each program transferred, with the Emulation option selected, which

will automatically LOAD and RUN from diskette the emulation programs. These new program lines are listed below.

```
0 B=PEEK(788):POKE 37287,0
3 IFB<>45ANDA<3THENA=3:LOAD"EMULATEB",8,1
4 IFB<>45ANDA=3THENA=4:LOAD"EMULATEA",8,1
5 SYS49152
```

b. Transfer Considerations The addition of these lines to the Basic program during transfer with Emulation option selected require that the original Applesoft program does not contain the above line numbers. Otherwise the resultant program will contain lines with the same line numbers. The transfer program will not give an error condition if this occurs. The emulation lines have the advantage of making the resultant program a stand alone program, however, the LOADing of the emulation programs from the diskette will cause some delay every time the program is RUN.

c. Editing and LISTing Selecting the emulation option during transfer mode can cause other problems that must be considered. One problem occurs in the final manual editing of the program. Editing and LISTing of a program with new commands (emulation option selected during transfer) must be done while emulation programs are in memory. Programs edited with the new commands will not

tokenize the Basic keyword properly if the emulation programs are not installed in memory during the editing and LISTing process. The MENU program will allow the operator to install emulation programs for editing and LISTing. See Emulation mode chapters for more details.

d. Emulation Tradeoffs Another consideration that needs to be taken into account before selecting the Emulation option during transfer is in the design of the emulation software itself. The emulation programs inherently have some tradeoff limitations due to hardware differences between the C-64 and the Apple computer. A prime example of a design tradeoff is the Applesoft command HGR. On the Apple computer the HGR command will display the hires screen page 1 located at memory location $4000 hex (16384 decimal) and four lines of the TEXT screen using built in hardware. On the C-64 computer the emulated HGR command will display the hires screen page 1 located at memory location $D000 hex (53248 decimal) and four lines of the TEXT screen using interrupts to split the screen display. The results of the command are very similar graphically but, obviously, not exactly the same results occur. The differences due to emulation programming tradeoffs are further detailed in the Emulation Mode chapters. The emulation command

differences must be evaluated for each application program considered for transferring to the C-64.

In most applications these inconveniences are minor compared to the monumental task of generating the machine language routines that are needed to generated Applesoft commands not available on the C-64.

The main disadvantage of the Emulation option is a decrease in available memory for Basic programming. The Emulation option decreases available memory by 3840 bytes to a value of 35,072 bytes. See memory maps in Appendix C. Another problem with Emulation option is that the emulation software may interfere with other C-64 binary programs. The emulation software uses RAM memory from $9100 hex to $A000 hex and $C000 thru $D000 hex. These areas of RAM memory are commonly used by other machine language programs. The Dos C-64 Wedge program is a prime example of a program that uses the same area of RAM memory as the emulation software. The emulation software will not be compatible with any machine language program that uses the same areas of RAM memory. The machine language programs that do not reside in the area of RAM memory used by the emulation software may or may not work with the emulation software. The user will have to try each individual program to see if the machine language program is compatible with the emulation software.

Another problem is that many Applesoft commands are not directly compatible because of machine hardware differences (i.e. I/O commands, error commands etc.). Therefore some of the Applesoft commands are not implemented. Table XIX lists the unimplemented Applesoft commands.

TABLE XIX

APPLESOFT COMMANDS NOT IMPLEMENTED BY EMULATION SOFTWARE

DRAW N AT C,R	ROT=
GR	SCALE=
HLIN B,E AT R	SCRN
IN# X	SHLOAD
ONERR GOTO	STORE
PR# A	VLIN A,B AT C
RECALL	XDRAW A AT C,R
RESUME	COLOR=
PLOT	AT
DEL	

5. <u>Apple Character Set Option</u>

The Apple character set option allows the user to create the Apple II character set on the C-64. This new character set located under Basic Rom ($A000 to $C000) might be needed with certain application programs under consideration for transfer. This option relocates the Basic Text screen from 1024 decimal to 35840, which

severely limits RAM memory available for program use. The Basic end of RAM pointer must be moved below the new TEXT screen location. The available free RAM memory for Basic programs use drops to 29,696 bytes. The normal available memory is 38911 bytes (i.e.after power up). See Appendix C memory maps for more details. Consequently this option should be selected only when the application program to be transferred dictates the use of the Apple II character set. Similar to the Emulation option, the Apple II character set option will add additional lines to the program that will allow the application program to install the Apple II character set. These lines are shown below.

```
0 B=PEEK(788):POKE37287,0
1 IFA=0THENA=1:LOAD"CHARACTERSET",8,1
2 IFA=1THENA=2:SYS36880
3 IFB<>45ANDA<3THENA=3:LOAD"EMULATEB",8,1
4 IFB<>45ANDA=3THENA=4:LOAD"EMULATEA",8,1
5 SYS49152
```

Note that selecting the Apple character set will automatically invoke the Emulation option. See chapters on Emulation option for further details.

B. Emulation Mode

1. Description and Operation

The Emulation mode of operation can be entered in

two ways: by selecting the option from the MENU program, or by running a program that has been transferred using the Emulation option. Either method achieves the same result; Emulation mode is entered and the C-64 Basic operating system is enhanced. This section will describe in detail these enhancements.

When Emulation mode is entered the displayed TEXT screen will clear to black. At this time a Run/Stop-Restore key sequence can be used to determine if Emulation mode is still in effect. If the display screen stays black then Emulation programs are intact and Emulation is running. If the display screen turns Blue then Emulation programs are not installed and a warm start return to normal C-64 Basic has occurred. Emulation mode may be exited by issuing a KILL command which will detach the emulation programs. If Emulation programs have been detached by the KILL command and the programs have not been corrupted the operator may reconnect Emulation mode by the command SYS 49152. Otherwise, normal entry (described above) must be used to install Emulation programs.

When the Emulation mode is entered, the emulation software will reset the Basic end of RAM pointer to protect the Emulation software. The end of RAM pointer is

normally set at $9100 hex but is moved to $8C00 hex when the Apple Character Set is installed. This Basic end of RAM pointer is changed during the execution of the Basic HIMEM: command. The value of the HIMEM variable must not be changed to exceed these values or Basic will destroy the emulation software.

2. Reason For Use Of

The Emulation mode of operation is used for four basic reasons listed in Table XX.

TABLE XX

FOUR REASONS TO USE EMULATION MODE

1.) Editing a program which includes Emulation commands listed in Table # 6.

2.) LISTing a program with Emulation commands incorporated into it.

3.) Running a program with Emulation commands incorporated into it.

4.) Execution of new Emulation commands in direct mode.

3. Compatibility of Commands

All four reasons involve a common element, the use of "tokens" to create new Basic commands not normally resident in C-64 Basic operating systems. Nevertheless,

emulation programming cannot make the host computer (C-64) program commands exactly like the Apple computer commands in all instances because of hardware design differences, limited memory, and other differences in the operating system design philosophies. This section will explain these differences.

a. Directly Compatible Implemented Commands The Applesoft commands listed in Table XXI can be assumed to be compatible emulated commands with no discernible differences between C-64 and Applesoft commands.

TABLE XXI

APPLESOFT AND EMULATED C-64 COMMAND EQUIVALENTS

HIMEM:	NOTRACE
HOME	POP
HTAB	SPEED=
VTAB	TEXT
INVERSE	TRACE
LOMEM:	HCOLOR=
NORMAL	

The differences in Apple commands and emulated C-64 commands can be grouped into four different classification areas as follows: disk commands, graphics commands, I/O commands and general commands (miscellaneous). The commands listed below in Table XXII are different in some degree from their Apple equivalents.

TABLE XXII

APPLE AND EMULATED C-64 COMMANDS THAT ARE DIFFERENT

Disk	Graphics	I/O	General
BLOAD	HGR2	GET(GIT)	FLASH
BSAVE	HGR	PDL	
EXEC	HPLOT		
BRUN			
CATALOG			

b. <u>Input/Output Commands</u> The Emulation mode GIT (GET) command tries to eliminate some of the dissimilarities between the Applesoft GET command and the C-64 GET command. The main reason for implementation of the GIT command is so that the C-64 will wait for a keystroke. And thus multiple variable operands may be successfully used on the C-64. The C-64 GET command usually returns a null or zero in most case of multiple operands. The GIT command will wait for a keystroke for each operand and return a value to all the operands in accordance with normal C-64 GET rules as described previously (see Traps and Pitfalls chapter). The transfer software will convert all Applesoft GET commands to the newly implemented GIT command if the Emulation option is selected.

The Applesoft PDL(n) command returns a value related to the position of the paddle. Where n is the

paddle number. n must be 0-3 since the Apple supports only four paddles. The value of PDL(n) is returned as an integer that ranges from 0 thru 150K.

The C-64 PDL(n) command will also return a value related to the position of the paddle, where n is the paddle number. However the C-64 values will differ from the Apple's values because of paddle hardware differences between the C-64 and the Apple. Thus, the programmer must modify the program to handle the new values from the C-64 paddle command PDL(n).

c. <u>Graphics Commands</u> The Applesoft Graphics commands HGR and HGR2 will display and clear to Black either of two high resolution screens (HiRes). HiRes screen page 1 at memory location $2000 to $3FFF hex is displayed and cleared by the HGR command. The HGR command also displays the bottom 4 lines of the text screen located at $400 hex. HiRes screen page 1 has a 280 horizontal dots by 160 vertical dots display. The HGR2 command will display and clear to black HiRes screen page 2 at memory location $4000 to $5FFF hex. HiRes screen page 2 has a 280 horizontal dots by 192 vertical dots display.[38]

The implemented C-64 HGR command will display and clear to black HiRes screen page 1 (Bit Mapped Mode) at

location $E000 to $FFFF (under Kernel ROM). The HiRes screen page 1 has a 320 horizontal dots by 168 vertical dots display. Also displayed by using Raster interrupts the bottom 4 lines of TEXT screen located at memory address $400 hex. The implemented C-64 HGR2 command has 320 horizontal dots by 200 vertical dots resolution and is located at $4000 to $5FFF. Color memory is located at $6400 hex and $CC00 hex for HGR2 and HGR respectively.[39] Disk command operations will temporarily disrupt the C-64 HGR screen due to the disabling of interrupt processing by the disk commands. Erratic operation might occur, thus, to be safe disk commands should not be done during HGR mode.

The Applesoft HPLOT h,v [TO h2,v2] command sets a point (turns a pixel on) or a series of points (a line) on the current HIRES screen (HGR or HGR2 screen). The color of the point is determined by the most recent HCOLOR command. The horizontal parameter range is 0 thru 279. The vertical parameter range is 0 thru 159 or 0 thru 191 for HGR or HGR2 respectively. On the Apple the background color (pixel off color) is always black and the foreground color (pixel on) is determined by the most recent HCOLOR parameter.[40]

The emulated C-64 HPLOT h,v [TO h2,v2] command sets a point (turns a pixel on) or a series of points (a line)

on the current HIRES screen. The color of the point(s) is determined by the most recent HCOLOR command. The horizontal parameter range is extended to 0 thru 319. The vertical parameter range is 0 thru 167 or 0 thru 199 for HGR or HGR2 respectively. On the C-64 the background color is cleared to Black and all pixels are turned off when the HGR(2) command is executed. However the user may change the background color (pixel off color) using the POKE command. See memory maps in Appendix C. The C-64 HCOLOR command sets the current foreground (pixel on color). The foreground and background colors are limited to a resolution of an 8 by 8 pixel square.[41] Thus when plotting a point using HPLOT, if a pixel is already on in this 8 by 8 color block, it will change color to the color currently being plotted. The resulting appearance will then be different from that of the Apple program. See memory maps in Appendix C for location of HIRES screens and their color memory locations.

The Applesoft HCOLOR= command and the implemented C-64 HCOLOR= command set the color of the next line to be drawn by the HPLOT command. The colors specified are listed in Table XXIII.[42]

TABLE XXIII

COLOR CODES FOR HCOLOR COMMAND

EXP. HCOLOR= N   N RANGES FROM 0-7

COLOR	COLOR CODE
BLACK	0
GREEN	1
VIOLET	2
WHITE	3
BLACK	4
ORANGE	5
BLUE	6
WHITE	7

d. General Commands   The Applesoft FLASH command will cause subsequent PRINT commands to print FLASHing characters on the TEXT screen. This is done by storing a FLASHing character code to TEXT screen memory. The Apple hardware will recognize the FLASH code and will cause the character to switch from an INVERSE character to a NORMAL character at a steady rate. This results in the blinking of the character on the TEXT screen. The implemented C-64 FLASH command will also cause subsequent PRINT commands to print FLASHing characters on the TEXT screen. The emulation software will switch the character from an INVERSE character to a NORMAL character using interrupt processing. During the PRINTing of the FLASHing characters, the flashing of the characters will accelerate temporarily until all characters have been PRINTed. Disk

operations will cause temporary suspension of the FLASHing characters due to the disabling of interrupt processing during the I/O operations of the disk commands.

e. Disk Commands The Apple disk commands implemented by Emulation Mode are listed in Table XXIV.

TABLE XXIV

EMULATED APPLE DISK COMMANDS

BLOAD	CATALOG
BRUN	EXEC
BSAVE	

All Apple disk commands that require a filename may have a filename with a length up to 31 characters. Apple arguments Slot, Volume, and Drive are optional. These arguments specify where the file is located. All arguments will be in hexadecimal or decimal format. All hex values will begin with a dollar sign. See Appendix B for further details.[43]

All the implemented C-64 disk commands will ignore optional arguments Slot and Volume. The Drive argument will be converted from drive 1 (Apple) to device 8 (C-64) and drive 2 (Apple) to device 9 (C-64). The drive argument will default to device 8. The filename used

with all emulated disk commands must not contain any embedded keywords (i.e. ON, AND, etc.). If the filename does contain any embedded keywords the embedded keyword will be "tokenized" to one character with unpredictable filename results. The C-64 filename must not be greater than 16 characters. The emulated disk commands do not display any drive errors but do terminate the command upon drive errors. The emulated C-64 disk commands are not to be embedded in a PRINT statement, unlike Apple II which requires a PRINT statement for Apple Dos commands in a program.

The Apple BLOAD command will load a binary file into Apple's memory from diskette. The BLOAD command requires only the filename of a binary file. Types of files other than binary files will give file type error messages. The BLOAD command will accept a load address argument to load the file at a specific memory address.[44]

The emulated C-64 BLOAD command will load any "PRG" file Binary or Basic. The C-64 BLOAD command will accept a load address in hex or decimal using the Apple's criteria of a dollar sign beginning any hex value. See Appendix B for further details.

The emulated C-64 BSAVE disk command will not overwrite an already existing file and a drive error will occur, unlike the Apple II BSAVE command which will

overwrite an existing file. The C-64 BSAVE command will save binary data from address A with a length L to the specified file on diskette similar to the Apple BSAVE command.

The emulated C-64 BRUN command will BLOAD the specified "PRG" file then do a machine language jump (SYS) to the start address of the file. If no address is specified the jump will be to the memory address where the file was BSAVEd (default). Warning: Do not BRUN Basic (PRG) files or the computer will crash.

The emulated C-64 CATALOG command will display the directory of device 8 only. Any Slot and Drive arguments will cause a syntax error to occur. Any Drive errors will cause aborting of the command and no error message will be displayed.

The emulated C-64 EXEC command will execute Basic command lines from the specified disk file. Every command line in the specified file must be terminated with a return and be less than 80 characters in length. All commands must be direct commands (i.e. GET will not work). See Appendix B for further details on any Apple disk command.

APPENDIX A

ALPHABETIC LISTING OF BASIC KEY WORDS OF APPLE COMMANDS

This appendix will allow the user to determine the command compatibility of the Apple and Commodore 64 commands. The Apple command is listed followed by a brief description of the command. Following this will be the differences/similarities with the Commodore 64 command. The Commodore 64 (C-64) command will further be divided, if necessary, to show any differences between the commands when the emulation program is selected by the user. An asterisk (*) in front of the command indicates possible compatibility problems between Apple and C-64 versions.[45,46]

ABS(X)

Apple : returns the absolute value of an expression.

C-64 command: Same as Applesoft.

*AND

Apple : A AND B a logical operator that returns a True(1) or False(0) value based on a binary computation. If A or B is zero then the ANDed result is zero.

C-64 command: AND a bitwise logical operator that returns a true bitwise binary AND value of A and B. When using

AND with True/False evaluations the computer assigns a value of (-1) if the expression is True (a non-zero bitwise binary AND of A and B) or a value of (0) to the expression when False (a zero bitwise binary AND of A and B) when used in a comparison test.

*APPEND

Apple : in the form 'PRINT D$;"APPEND FILENAME"' : Opens a sequential file and positions the write pointer at the end of file.

C-64 command: OPEN N,DV,SA,"FILENAME,A" where N is the logical file number; DV is the device number (usually 8); SA is the secondary address; Filename is the name of file.

ASC(N$)

Apple : returns ASCII value of first character of N$. This is inverse of CHR$.

C-64 command: Same as Applesoft

ATN(X)

Apple : returns angle (in radians) whose tangent is x.

C-64 command: Same as Applesoft.

*BLOAD

Apple : in the form 'PRINT D$;"BLOAD FILENAME,AN,SM,DO,VP"' where filename is a binary file; N is memory location where the file is to be loaded. If N

is omitted file is loaded at address from which it was BSAVED.; M is slot number of disk drive controller.; O is desired drive number.; P is the volume number of the disk.
C-64 command: LOAD "filename"8,1 loads binary file at saved address. This command causes Basic program to restart.
C-64 command with emulation: BLOAD filename,AN,SM,DO,VP where filename is a binary file; N is memory location where file is to be loaded. If omitted file is loaded at saved address.; M and P is ignored; O is 1 or 2 and selects disk drive number 8 or 9.

*BSAVE
Apple : in the form 'PRINT D$;"BSAVE FILENAME,AN,LQ,SM,DO,VP"' where filename is a binary file with first byte at memory location N with Q bytes in length. See BLOAD for description of other parameters.
C-64 command: Not available
C-64 command with emulation: BSAVE filename ,AN,LQ,SM,DO, VP where filename is a binary file to be saved. The file's first byte will start at N and be Q bytes in length.See BLOAD for description of other parameters.

*CALL N
Apple : causes execution of a machine language routine at

memory location N. N ranges from -65535 to 65535.

C-64 command: SYS N causes execution of a machine language routine at memory location N. N ranges from 0 to 65535.

Note: Apple machine language programs generally will not execute properly on the C-64.

*CATALOG ,SM,DO

Apple : in the form: 'PRINT D$;"CATALOG,SM,DO"' will display directory of disk drive number O with drive controller in M slot.

C-64 command: LOAD "$",8 will load directory into Basic programming space and destroy program.

C-64 command with emulation: CATALOG ,SM,D0 will display directory of drive O. If O is 0 then first drive is selected (usually 8). If O is a one then second drive (usually 9) is selected. M parameter is ignored.

*CHR$(N)

Apple : returns the character represented by the ASCII code N.

C-64 command: CHR$(N) command has some non-standard CHR$ codes. See C-64 Programmers Reference manual.

CLEAR

Apple : clears Basic variables.

C-64 command: CLR clears Basic variables.

*CLOSE

Apple : in the form 'PRINT D$;"CLOSE FILENAME"' closes the sequential file "Filename"

C-64 command: CLOSE N where N is the logical file number of file previously OPENed. Note: Transfer program assumes only one sequential file active at any given time and assigns logical file number 14 to it.

*COLOR=N

Apple : sets the color for plotting in low-resolution graphics. N ranges from 0 to 15. When in TEXT mode COLOR determines which character is affected by PLOT command.

C-64 command: Not available.

COS(X)

Apple : returns the cosine of angle X. X is in radians.

C-64 command: Same as Applesoft.

DATA A,A,...

Apple : where A is a constant to used by the READ command.

C-64 command: Same as Applesoft.

DEF FN A(X)=E

Apple : gives a user defined function named A. X is a numeric variable that is passed to the function. E is a numeric expression that usually includes X as a variable.

C-64 command: Same as Applesoft.

DIM arrayname (A,B,..)

Apple : specifies the dimensions (length) of numeric or string arrays.

C-64 command: Same as Applesoft.

*DRAW N at C,R

Apple : places a shape on the screen, where N specifies a shape in shape table in memory. C is column and R is row where the shape is to be drawn.

C-64 command: Not available.

END

Apple : terminates program execution and closes all files.

C-64 command: Same as Applesoft.

*EXEC FILENAME

Apple : executes the batch (sequential) file filename residing on disk at the end of the program. After statement containing the command all input comes from the file and not the keyboard. Also all immediate DOS commands entered are executed.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft, except no command lines longer than 80 characters are allowed and the DOS commands are not immediately executed.

EXP(X)

Apple : returns the number e raised to the X power.

C-64 command: Same as Applesoft.

*FLASH

Apple : causes output display to cycle between INVERSE and NORMAL.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft. This command uses interrupt proccessing and will not flash during I/O commands.

FN N(E)

Apple : where N is a variable name and E is a expression to be used by the function. See DEF FN.

C-64 command:Same as Applesoft.

FOR Var=A1 TO A2 STEP C

Apple : this command executes a loop terminated by a NEXT command. The loop is executed from Var equals A1 to A2 incremented by C each time thru loop.

C-64 command: Same as Applesoft.

FRE(X)

Apple : will give amount of free memory available for program.

C-64 command: Same as Applesoft.

*GET A$,B$,...

Apple : retrieves a single character for each variable name listed (i.e. A$,B$,..) from current input device. The program is halted until all variables are filled. No prompt is displayed and no RETURN key is needed.

C-64 command: GET A$,B$,... retrieves a character from keyboard for each variable listed. The program is not halted and null character is returned if a key is not pressed.

C-64 command with emulation: GIT A$,B$,.. Similiar to Applesoft. See Emulation Mode Chapters.

GOSUB N

Apple : executes a subroutine at line numbered N and returns to the subsequent line after encountering a RETURN command in the subroutine.

C-64 command: Same as Applesoft.

GOTO N

Apple : causes program to jump to line numbered N and continue execution of program.

C-64 command: Same as Applesoft.

*GR

Apple : causes the low resolution graphics screen to be

displayed.

C-64 command: Not available.

*HCOLOR=N

Apple : sets the color to be used by HPLOT in high resolution graphics mode. N ranges from 0 to 7.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

*HGR

Apple : causes the computer to display the high resolution graphics screen page 1 leaving four lines of TEXT screen at bottom of screen. The graphics screen is cleared to Black.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft. This command is implemented using the interrupt system to split the screens. Do not try to do I/O during this mode.

*HGR2

Apple : causes the computer to display the high resolution graphics screen page 2. The screen is cleared to Black and no TEXT screen lines are displayed.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

*HIMEM:M

Apple : will reserve memory space above location M. M is the upper memory location for the Basic program and variable storage.

C-64 command: Top of Basic Memory may be set by POKE 51,LOW:POKE 52,HIGH:POKE55,LOW:POKE56,HIGH:CLR. Where LOW is the least significant hexadecimal byte and High is most significant hexadecimal byte of memory location M.

C-64 command with emulation: Same as Applesoft.

*HLIN B,E at R

Apple : where B is the beginning column and E is ending column, and R is the row at which a horizontal line is plotted in low resolution graphics screen. HLIN used in TEXT mode causes a line of characters to be displayed.

C-64 command: Not available.

*HOME

Apple : clears the text window and moves the cursor to the upper left corner. The text window may or may not be the entire screen.

C-64 command: PRINT CHR$(147); clears the entire screen and moves cursor to upper left corner. C-64 normally has no window capabilities.

C-64 command with emulation: Home clears the entire text

screen and moves the cursor to the upper left corner.

*HPLOT H,V or HPLOT H,V TO H1,V1 TO ...

Apple : causes a point or series of lines to be plotted on high resolution graphics screen. H is horizontal coordinate (0-279) and V is vertical coordinate (0-191). The color of the line is determined by the most recent HCOLOR command.

C-64 command: Not available.

C-64 command with emulation: HPLOT H,V or HPLOT H,V TO H1,V1 TO... causes a point or series of lines to be plotted on high resolution graphics screen. H is horizontal coordinate (0-319) and V is vertical coordinate (0-199). The color of the line is determined by the most recent HCOLOR command.

*HTAB N

Apple : position cursor horizontally at point N (0-255) positions from beginning of current line position.

C-64 command: Not available, but can be done with pokes.

C-64 command with emulation: Same as Applesoft.

IF A THEN B or IF A GOTO B

Apple : causes computer to execute instruction B or jump to line B if expression A is true. If A is false does nothing and continue to next program line.

C-64 command: Same as Applesoft.

*IN#X

Apple : redirects input to come from slot specified by X.

C-64 command: Not available.

*INPUT "prompt"; A,B

Apple : causes prompt message to be displayed, program halted for a response from keyboard followed by a return key. All response characters will be displayed. The prompt message is optional and computer will display a question mark if omitted. If RETURN is pressed Apple returns a null or 0.

C-64 command: Same as Applesoft,excepts returns old value if RETURN is pressed. See Traps and Pitfalls section.

INT(N)

Apple : will convert N a real number to an integer by truncating the fractional part of N.

C-64 command: Same as Applesoft.

*INVERSE

Apple : causes all output printed to screen to be in inverse color.

C-64 command: PRINT CHR$(18) will invoke inverse mode. Printing a return character will terminate inverse mode.

C-64 command with emulation: Same as Applesoft.

LEFT$(X$,N)

Apple : returns the left most N characters of string X$. N ranges from 1 to length of X$.

C-64 command: Same as Applesoft.

LEN(X$)

Apple : returns the length of string X$.

C-64 command: Same as Applesoft.

LET N=X

Apple : assigns the value of X to the variable N.

C-64 command: Same as Applesoft.

*LIST X-Y or LIST X,Y

Apple : list the program to the screen from line numbers X to Y. X and Y are optional.

C-64 command: LIST X-Y, list the program to the screen from line numbers X to Y. X and Y are optional. If used within a program aborts the program after listing.

*LOAD FILENAME

Apple : loads the Basic file into memory. All variables are cleared and data files are closed. After loading, Basic returns to command mode.

C-64 command: LOAD "FILENAME",DV,LOC loads a file into memory, and closes all files. However does not clear

variables or reset Basic memory pointers. After load is complete it automatically runs the Basic program. Dv is the device from which to get the file. LOC specifies the area at which to load the file. Zero is Basic default area. One loads the file to area from which it was saved.

LOG(x)

Apple : returns the natural logarithm of X. Inverse of EXP(x).

C-64 command: Same as Applesoft.

*LOMEM:X

Apple : set the lowest memory location available for variable storage. X is a valid memory location.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

MID$(X$,A,B)

Apple : returns the portion of string X$ starting at position A and B characters in length. If B is omitted then returns the string to the right of position A of X$.

C-64 command: Same as Applesoft.

NEW

Apple : deletes the Basic program currently in memory and clears all variables and returns control to the command

mode.

C-64 command: Same as Applesoft.

*NORMAL

Apple : restores INVERSE or FLASH to the normal TEXT mode.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

*NOT A

Apple : will return a 1 if A is false or a 0 if A is true. A is any valid expression.

C-64 command: NOT A produces an integer ones-complement by complementing the value of each bit. If A is a real number, A is converted to an integer then complemented. NOT can also be used in a comparison to reverse the true/false value which was the result of a relationship test returning a True (a non-zero ones-complement value) or a False (0). See Traps and Pitfalls section for more detail.

*NOTRACE

Apple : cancels the effects of TRACE command.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

ONERR GOTO N

Apple : when an error is encountered after this statement

program execution will jump to the routine beginning at line N, and continue until the word RESUME is encountered.
C-64 command: Not available.

*ON V GOSUB A,B
Apple : causes conditional program execution of GOSUB command depending on the value of V a numeric expression. If V is 0 or a number greater than the number of lines listed, then the program does not branch but falls through to next line, otherwise it branches then returns to the next line following the current line. A negative value of V will give an error condition.
C-64 command: Same as APPLESOFT, except if expression is True (1 for APPLESOFT, -1 for C-64) an error could result. The expression below would branch for APPLESOFT but give and error on the C-64).
EX. ON (0<1) GOSUB 100

*ON V GOTO A,B
Apple : same as ON V GOSUB A,B except it causes the computer to execute a GOTO command and thus does not return to the line following the current line.
C-64 command: Also REF. ON V GOSUB A,B

*A OR B
Apple : is a logical operator and returns a True (1) if

either of the values of expressions A or B is true or non-zero, otherwise returns a False (0).

C-64 command: A OR B is a bitwise logical bitwise operator that returns a bitwise binary OR value of A and B. It allows an evaluation of two item A and B returning a True a value of -1 (a non-zero bitwise binary OR value) or a False a value of 0 (a zero bitwise binary OR value).

*PDL(N)

Apple : reads the paddle N where N is 0-3. PDL will range from 0 to 255.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

*PEEK(M)

Apple : returns the contents of memory location M.

C-64 command: Same as Applesoft. Many memory location of the C-64 do not contain the same value, as the APPLE.

*PLOT H,V

Apple : places a dot at H the horizontal coordinate (range 0-39)and V the vertical coordinate on the low-resolution graphics screen (range 0-47) or TEXT screen (range 0-39).

C-64 command: Not available.

*POKE M,N

Apple : stores the value of N into memory location M. N must be between 0 and 255.

C-64 command: Same as Applesoft,except many memory locations in C-64 do not have the same function as the APPLE.

*POP

Apple : causes the most recent RETURN address to be deleted from the top of the return address stack.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

POS(N)

Apple : returns the current horizontal cursor position.

C-64 command: Same as Applesoft.

*PR# N

Apple : redirects output commands (PRINT,LIST,etc.) to specified slot N. N ranges from 1-7

C-64 command: CMD N redirect output to the file number N. The file must have been previously OPENed.

*PRINT

Apple : list where list is a expression or combinations of variables, strings to be printed to current output device. Also used to send disk commands to disk drive in the form "PRINT D$" where D$=CHR$(4)

C-64 command: PRINT list where list is a expression or combinations of variables, strings to be printed to current output device. When numeric values are printed, leading blanks may be inserted.

READ A,B,...

Apple : reads data from DATA statements and assign the data values to variables A and B. A and B can be string data or numeric data.

C-64 command: Same as Applesoft.

*RECALL N

Apple : reads values from the cassette into array N.

C-64 command: Not available.

REM

Apple : used to add comment lines to the program. REM statements are not executed.

C-64 command: Same as Applesoft.

RESTORE

Apple : resets the DATA pointer to the first Data statement.

C-64 command: Same as Applesoft.

*RESUME

Apple : returns control from an error-handling routine to

the statement that caused the error.

C-64 command: Not available.

RETURN

Apple : ends a subroutine and causes program execution to return to the line following the calling GOSUB line.

C-64 command: Same as Applesoft.

RIGHT$ (X$,N)

Apple : returns a string which consist of the right N characters of string X$.

C-64 command: Same as Applesoft.

RND(N)

Apple : returns a pseudo-random number between 0-1 if N is positive. If N is zero will give same number each time. If N is negative the value of N will act as a seed for a new random number sequence.

C-64 command: Same as Applesoft.

*ROT=N

Apple : will determine the amount of rotation of a high resolution graphics shape before DRAWing on the screen. If N is 1 represents 1/64 of a circle rotation.

C-64 command: Not available.

*RUN,RUN N,RUN FILENAME

Apple : will begin executing a program in memory at lowest line number or at line numbered N. The last form will load and RUN program file named FILENAME.

C-64 command: RUN,RUN N will begin executing a program in memory at lowest line number or at specified line numbered N. RUN FILENAME is not available. Refer to LOAD command.

*SAVE

Apple : in the form PRINT D$;"SAVE FILENAME" ,SA,DB,VC saves the Basic program currently in memory. Parameter SA specifies the slot A, DB specifies the drive B, and VC specifies the volume.

C-64 command: SAVE "FILENAME",A saves the current program to device A. (A=1 cassette, A=8 diskette)

*SCALE=N

Apple : sets the scale factor for shapes drawn from high resolution shape table. This command will expand original size by N times.

C-64 command: Not available.

*SCRN (C,R)

Apple : return the color of the low-resolution graphics screen at column C and row R.

C-64 command: Not available.

SGN(A)

Apple : returns the sign of expression A. If A is negative returns a -1, If A is zero returns a zero or if A is positive returns a 1.

C-64 command: Same as Applesoft.

*SHLOAD

Apple : loads a shape table from cassette.

C-64 command: Not available.

SIN(A)

Apple : will give the trigonometric sine of A in radians.

C-64 command: Same as Applesoft.

SPC(N)

Apple : in the form "PRINT SPC(N)" will print N number of spaces

C-64 command: Same as Applesoft.

*SPEED=N

Apple : sets the speed of outputting data where N is the speed of output. The slowest speed is zero and the fastest (default) is 255.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

SQR(N)

Apple : returns the square root of N.

C-64 command: Same as Applesoft.

STEP N

Apple : sets the incremental value of a loop. See FOR command.

C-64 command: Same as Applesoft.

STOP

Apple : ceases execution of program and returns to command level.

C-64 command: Same as Applesoft.

*STORE N

Apple : where N is any valid numeric array name will cause the array elements to be stored on disk. See RECALL.

C-64 command: Not available.

STR$(N)

Apple : converts the numeric expression N to a string representation.

C-64 command: Same as Applesoft.

TAB(N)

Apple : tabs to horizontal position N. TAB(N) is used only in PRINT statements.

C-64 command: Same as Applesoft.

TAN(A)

Apple : returns the trigonometric tangent of N in radians.

C-64 command: Same as Applesoft.

*TEXT

Apple : switches the display to the normal TEXT screen and moves cursor to HOME position.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

*TRACE

Apple : turns on trace utility which displays line numbers of line currently being executed. This command is usually used for debugging programs.

C-64 command: Not available.

C-64 command with emulation: Same as Applesoft.

*USR(X)

Apple : calls a machine language program and passes value A to it.

C-64 command: Same as APPLESOFT.

NOTE: Many machine language programs are not compatible with different computer systems.

VAL(A$)

Apple : convert string A$, a string expression of a number, to a numeric variable. If A$ is not numeric a value of zero is returned.

C-64 command: Same as Applesoft.

*VLIN A,B AT C

Apple : draws a vertical line from row A to row B at column C on the low resolution screen.

C-64 command: Not available.

*VTAB N

Apple : moves the cursor to line N.

C-64 command: Not available, but can be done with pokes.

C-64 command with emulation: Same as Applesoft.

*WAIT A,B,C

Apple : halts program execution while monitoring the condition of memory address A. The value at address A is exclusively ORed with C, then result is ANDed with memory address A value. The program waits for the value of this complex operations to become non-zero value. A may be an integer number between -65535 and 65535.

C-64 command: WAIT A,B,C same function as APPLESOFT, except A ranges from 0 thru 65535.

*XDRAW A AT C,R

Apple : draws the shape A from the shape table currently in memory at row R and column C on the high resolution screen.

C-64 command: Not available.

APPENDIX B

ALPHABETIC LISTING OF APPLE DISK COMMANDS

This appendix will allow the user to determine the command compatibility of the Apple and Commodore 64 disk commands. The Apple disk command is listed followed by a brief description of the command. Following this will be the differences/similarities with the Commodore 64 disk command. The Commodore 64 (C-64) disk command will further be divided, if necessary, to shown any differences between the commands when the emulation program is selected by the programmer.

The Apple Dos 3.3 disk commands have different syntax in program mode or immediate execution mode. This appendix will deal with only the program mode syntax.

Each of the Apple Disk commands has similar variables that are optional or required. The variables will have brackets enclosing them in the command line to show they are optional. All required variables will be shown without brackets enclosing them. The value of a variable must begin with a capital letter depicting the variable type. All variables must be separated by commas. The following section will describe each variable type.

The individual disk commands will detail which variables are optional or required. The disk command variables are listed below.

VARIABLE TYPE AND VALUE DESCRIPTION

Ss - The slot number of disk controller, usually 6. The variable s must be in the range 1 through 7.

Vv - The volume number of diskette. The variable v must be in the range 0 through 254.

Dd - The drive number. The variable d is either 1 or 2. The C-64 emulation software will assign device 8 when d is 1 or device 9 when d is 2.

Rp - The position number. The relative position of the read/write pointer in a sequential value. The p parameter will move the read/write pointer to the p-th field following current file position . The variable p must be in the range 0 through 32767.

Rr - The record number in random access file to position the read/write pointer. The variable r must be in the range 0 through 32767.

Aa - The memory address in RAM from which a program will be BSAVEd from or BLOADed to. The variable a must be in the range 0 thru 65535.

Bb - The byte number to position the

read/write pointer in a sequential file or the byte number to position the read/write pointer within a record of a random access file.

Lj - The length specifier of a binary file. The variable j must be in the range 0 through 32767.

The value of any variable may be in hex or decimal. Hex values must begin with a dollar sign. The dollar sign must immediately follow the type specifying letter.

Most Apple Dos 3.3 commands require a filename. The filename may have up to 31 characters. Most C-64 emulated Apple disk commands also require a filename. The C-64 filename may have up to 16 characters. The C-64 filename must not include embedded Basic keywords. If embedded keywords are in the filename they will be tokenized to unrecognizable characters with unpredictable results for the filename. All C-64 emulated disk commands ignore the slot and volume variables. The drive variables values of 1 and 2 convert to device numbers 8 and 9 respectively.[47,48]

APPEND

Apple: in the form PRINT D$; "APPEND FILENAME" [,Ss] [,Vv] [,Dd]. The APPEND command opens an existing sequential

text file and sets the Write pointer at the end of the file. The transfer software will comment out (REM) the APPEND command during transfer.

C-64 command: in the form OPEN N,DV,SA,"FILENAME,A". This variation of the OPEN command OPENs the sequential file and sets the write pointer to the end of the file. N is the logical file number which ranges from 1 to 255. DV is the device number usually a value of or 9. SA is the secondary address or channel number which ranges from 1 to 15.

BLOAD

Apple: in the form PRINT D$; "BLOAD FILENAME" [,Aa] [,Ss] [,Dd] [,Vv]. The BLOAD command will load a binary file into the same RAM memory location from which it was BSAVEd or at starting RAM memory address a.

C-64 command: in the form LOAD "FILENAME",8,1. This variation of the LOAD command loads a "PRG" file at the saved address. This command causes the Basic program to restart.

C-64 command with emulation: in the form BLOAD FILENAME [,Aa] [,Ss] [,Vv] [,Dd]. The BLOAD command will load a "PRG" file into the same memory locations from which the file was BSAVEd or at starting address a.

BRUN

Apple: in the form PRINT D$; "BRUN FILENAME" [,Aa] [,Ss] [,Vv] [,Dd]. The BRUN command BLOADs the specified binary file to memory at BSAVEd address or memory address a then executes the program starting at the first memory location.

C-64 command: Not available.

C-64 command with emulation: in the form BRUN FILENAME [,Aa] [,Ss] [,Dd] [,Vv]. The BRUN command will do a emulated C-64 BLOAD then do a C-64 Basic SYS command at first memory location of he loaded file.

BSAVE

Apple: in the form PRINT D$; "BSAVE FILENAME" ,Aa ,Lj [,Ss] [,Vv] [,Dd]. The BSAVE command stores the contents of j memory bytes starting at location a on the diskette as the specified binary file, overwriting the file if it exists.

C-64 command: not available.

C-64 command with emulation: in the form BSAVE FILENAME ,Aa ,Lj [,Ss] [,Vv] [,Dd]. Same as Apple DOS 3.3, except the command will not overwrite an existing file on diskette. The software will not display the error and the data will not be stored on diskette. The command will abort on drive errors, but if in a program the program will continue.

CATALOG

Apple: in the form PRINT D$; "CATALOG" [,Ss] [,Dd]. The CATALOG command displays the directory of the specified diskette.

C-64 command: Not available. The C-64 can load in the directory but the Basic program will be destroyed. LOAD "$",8 will load the directory of the diskette into Basic programming space.

C-64 command with emulation: in the form CATALOG. The CATALOG command displays the directory of device 8. The C-64 CATALOG will accept no arguments. If any exist a syntax error will occur.

CHAIN

Apple: Not used in Applesoft Basic programs.

C-64 command: in the form LOAD "FILENAME",8. This variation of the LOAD command will load and RUN a Basic program with present variable values intact.

CLOSE

Apple: in the form PRINT D$; "CLOSE FILENAME" [,Ss] [,Vv] [,Dd]. The CLOSE command deallocates the buffer that is associated with the specified file on the diskette. If a WRITE command is in effect all characters in the output buffer are sent to the specified file on the diskette.

C-64 command: in the form CLOSE N. The C-64 CLOSE command will close the file on diskette specified by the number N, updating the BAM and the directory. The transfer software will convert all Apple CLOSE commands to the C-64 CLOSE 14 command.

DELETE

Apple: in the form PRINT D$; "DELETE FILENAME" [,Ss] [,Vv] [,Dd]. The DELETE command will erase the specified file from the specified diskette.

C-64 command: in the form OPEN 15,DV,15,"S0:FILENAME". This variation of the OPEN command will erase the specified file on the diskette at device DV. The device number DV is either 8 or 9.

EXEC

Apple: in the form PRINT D$; "EXEC FILENAME" [,Rp] [,Ss] [,Vv] [,Dd]. The EXEC command will execute a series of direct execution Basic commands or DOS commands from the specified file. The Rp variable starts the execution at the p-th field in the file. The file must be a sequential file. If an EXEC command is issued from a Basic program, subsequent INPUT statements in the program will receive the data from the specified sequential file. The EXEC command will wait until the end of the program before

executing any commands from the specified file. If an EXEC command is already in effect and another EXEC command is encountered the present file will CLOSE and execution will continue from the new specified file.

C-64 command: Not available.

C-64 command with emulation: in the form EXEC filename [,Ss] [,Vv] [,Dd]. The C-64 EXEC command will operate similar to the Apple EXEC command. The C-64 EXEC command will only accept commands up to 80 characters in length. The C-64 EXEC uses file number 5 thus, if a CLOSE 5 is encountered the EXEC command will abort. If the Rp variable is specified a syntax error will occur.

FP

Apple: Not used in a program .

C-64 command: Not available.

IN#

Apple: in the form PRINT D$; "IN# N". IN# redirects input to come from the slot specified by N. N ranges from 1 thru 6.

C-64 command: Not available.

INIT

Apple: Not used in a program.

C-64 command: in the form OPEN 15,DV,15,"N0:NAME,N". This

variation of the OPEN command initializes (formats) the specified diskette at device DV. DV is the device number of 8 or 9. NAME is the desired name of the diskette. N is the diskette ID number which maybe any 2 character designators.

INT

Apple: Not used in a program.

C-64 command: Not available.

LOAD

Apple: in the form PRINT D$; "LOAD FILENAME" [,Ss] [,Vv] [,Dd]. The LOAD command LOADs the specified Basic program file into memory. All data files are CLOSEd. All Basic variables are CLEARed and Basic is returned to the command mode.

C-64 command: in the form LOAD "FILENAME",DV,N. If the LOAD command is issued from the keyboard, the Basic program will load into memory. If the LOAD command is issued from a Basic program it will load a Basic program into memory and RUN it; all Basic variables remain at their current state. If N is 0 then the file will load into Basic programming memory. If N is one, the file is loaded into memory from where it was SAVEd. DV is the device number which is usually a value of 8 or 9. See RUN command in Appendix B.

LOCK

Apple: in the form PRINT D$; "LOCK FILENAME" [,Ss] [,Vv] [,Dd]. The LOCK command will (by software) write protect the specified file from accidental deletion or change.

C-64 command: Not available.

MAXFILES

Apple: in the form PRINT D$; "MAXFILES N". The MAXFILES command sets the number of 595 byte buffers available for disk I/O. N is the number of buffers available for active files, from 1 to 16 (default is 3). The MAXFILES command must be used before any string variables are declared because it moves the HIMEM pointer. When used, the MAXFILES command is usually the first statement in the program.

C-64 command: Not available.

MON

Apple: in the form PRINT D$; "MON" [,C] [,I] [,O]. The MON command will allow the programmer to monitor the various disk I/O operations. The C parameter will allow all disk commands to be displayed on the screen or the current output device. The I parameter will allow all input information from the disk to the Apple to be

displayed on the current output device. The O parameter will display on the current output device all output information that is sent to the disk from the Apple.
C-64 command: Not available.

NOMON

Apple: in the form PRINT D$; "NOMON" [,C] [,I] [,O]. The NOMON command allows the programmer to disable all or part of the effects of the MON command. The C parameter will disable the display of all the disk commands. The I parameter will disable the display of all the Input information from the disk to the Apple. The O parameter will disable the display of all the output information being sent to the disk from the Apple. The Apple default state is NOMON C,I,O.
C-64 command: Not available.

OPEN

Apple: in the form PRINT D$; "OPEN FILENAME" [,Ss] [,Vv] [,Dd] for sequential files or in the form PRINT D$; "OPEN FILENAME" ,Lj [,Ss] [,Vv] [,Dd] for random access files. The OPEN command will OPEN the specified text file with a record length j. If j is not specified, a sequential file is OPENed. When a file is OPENed a memory buffer of 595 bytes is allocated to the specified text file. The read/write pointer is positioned to the beginning of the

specified file. If the specified file is already OPEN, this command CLOSEs the file before OPENing the file.
C-64 command: in the form OPEN N,DV,SA,"FILENAME,TYPE, MODE". The OPEN command will OPEN the specified type file with the specified mode of access. N is the logical file number which ranges from 1 to 255. DV is the peripheral device number, usually a value of 8 or 9. SA is the secondary address or command channel number which ranges from 1 to 15. Type is the type of file to be OPENed. Type is an S for a sequential file, an R for a relative file (random access) and default (no type specified) is a program (PRG) file. Mode is the direction of access. If MODE is an R the file is setup for reading, If MODE is a W the file is setup for writing, and if MODE is an A the file is setup for appending.

POSITION

Apple: in the form PRINT D$; "POSITION FILENAME" ,Rp. The POSITION command will position the read/write pointer to the beginning of the p-th field following the current location of the read/write pointer of the specified sequential file. Fields are terminated by a return character (i.e. ASCII code 13).
C-64 command: in the form PRINT# N,"P" CHR$(SA+96) CHR$(REC#LO) CHR$(REC#HI) CHR$(POS). Where N is the

logical file number of the disk command channel (usually 15). The record number (REC#) is in a 2 byte format as calculated below.

REC#HI = INT (REC#/256)

REC#LO = REC# - (REC#HI * 256)

The record number value (REC#) ranges from 1 to 720. POS is the position within the specified record where the write pointer is pointing. The POS values ranges from 1 to 254.

PR#

Apple: in the form PRINT D$; "PR# N". The PR# command will send subsequent outputs to slot N. If the disk controller card is installed in slot N, DOS is booted. The PR# command is usually used to send output to a printer (PR#1).

C-64 command: in the form CMD N. The CMD command redirects output to the logical file number N. The file must have previously been OPENed.

READ

Apple: in the form PRINT D$; "READ FILENAME" [,Rr] [,Bb]. The READ command will allow subsequent INPUT and GET commands to obtain their data from the specified file. If Rr parameter is specified the file is a random access file

and the read/write pointer is positioned to the r-th record. If the Bb parameter is specified the read/write pointer will move to the b-th byte of the specified record. If the Rr parameter is not specified the Bb parameter will move the read/write pointer to the b-th byte of the specified sequential file.

C-64 command: Not available. However, INPUT# N and GET# N commands can be used to obtain data from the logical file number N.

RENAME

Apple: in the form PRINT D$; "RENAME OLD FILENAME,NEW FILENAME" [,Ss] [,Vv] [,Dd]. The RENAME command changes the name of the specified file in the diskette directory to the new specified name.

C-64 command: in the form PRINT# N, "RENAME: NEWNAME= OLDNAME". Where N is the logical file number of the disk command channel (usually 15). The RENAME command will change the name of the file in the diskette directory.

RUN

Apple: in the form PRINT D$; "RUN FILENAME" [,Ss] [,Vv] [,Dd] within a program. The RUN command will LOAD the specified file from diskette into Apple memory, then RUN the program LOADed. See LOAD command in Appendix B and RUN command in Appendix A.

C-64 command: in the form LOAD "FILENAME",DV within a program. The LOAD command will load a file from diskette into C-64 memory then RUNs the program. The Basic variable values are not cleared. If using the LOAD command from a Basic program, be sure the program being LOADed is shorter than the current program in memory or the program will crash. DV is the device number, usually a value of 8 or 9.

If the RUN command is issued from the keyboard the program the program that is currently in memory is run. The RUN command clears all variables and starts at the beginning of the program.

SAVE

Apple: in the form PRINT D$; "SAVE FILENAME" [,Ss] [,Vv] [,Dd]. The SAVE command will store the current Basic program on diskette overwriting an existing file of the same name, if it exists. If the diskette contains a file with the same name but of a different file type a FILE TYPE MISMATCH error will be displayed.

C-64 command: in the form SAVE "FILENAME",DV. The SAVE command will store the the current Basic program on diskette. If a file with the same name exists on the diskette, the error light on the drive will flash and the program will not be stored on the diskette. A special

version of the SAVE command in the form SAVE "@0:FILENAME", DV maybe be used to overwrite an existing file on the diskette. This version is not recommended because of a software "bug" in the C-64 disk drive operating system. DV is the device number, usually a value of 8 or 9.

UNLOCK

Apple: in the form PRINT D$; "UNLOCK FILENAME" [,Ss] [,Vv] [,Dd]. The UNLOCK command will undo the software LOCK command and allow the software to change or delete the specified file.

C-64 command: Not available.

VERIFY

Apple: in the form PRINT D$; "VERIFY FILENAME" [,Ss] [,Vv] [,Dd]. The VERIFY command will check to see if a file is stored correctly on the diskette. The VERIFY command will check any type of file. The VERIFY command compares a new calculated checksum of all the data in each sector with the stored checksum; if they are equal no error message is printed.

C-64 command: in the form VERIFY "FILENAME",8. The VERIFY command will check to see if the current Basic program in the C-64 memory was stored correctly on the diskette. The

C-64 VERIFY command will work with only Basic "PRG" files.

WRITE

Apple: in the form PRINT D$; "WRITE FILENAME" [,Rr] [,Bb]. The WRITE command will allow subsequent PRINT commands to output data to the specified file. If the Rr parameter is specified the file is a random access file and the write pointer is positioned to the r-th record. If the Bb parameter is specified the write pointer is moved to the b-th byte of the specified record. If the Rr parameter is not specified, the file is a sequential file. If the Bb parameter is specified for a sequential file, the write pointer is set to the b-th byte of the sequential file.

C-64 command: Not available. However the PRINT# N command and CMD N can be used to output data to the logical file number N.

APPENDIX C

C-64 MEMORY MAPS

MEMORY ADDRESS

DECIMAL	HEX	
65535	FFFF	
		8K KERNAL ROM
57344	E000	
		4K I/O AREA
53248	D000	
		TRANSFER PROGRAM SECTION B
49152	C000	
		8K BASIC ROM
40960	A000	
		TRANSFER PROGRAM SECTION A
34048	8500	
		FREE RAM FOR BASIC PROGRAM STORAGE OR FOR BINARY PROGRAM STORAGE MAXIMUM LENGTH IS 32000 ($7D00 HEX) BYTES
2048	800	
		TEXT SCREEN
1024	400	
		RESERVED RAM FOR BASIC
32	20	

FIGURE 1 -- C-64 MEMORY MAP DURING TRANSFER MODE

MEMORY ADDRESS DECIMAL	HEX	
65535	FFFF	
		8K KERNAL ROM AND HIRES SCREEN I (UNDER ROM)
57344	E000	
		4K I/O AREA AND CHARACTER ROM
53248	D000	
		COLOR MEMORY FOR HIRES SCREEN I
52224	CC00	
		EMULATION PROGRAM AREA (PART B)
49152	C000	
		8K BASIC ROM AND NEW CHARACTER SET (UNDER ROM)
40960	A000	
		EMULATION PROGRAM AREA (PART A)
37120	9100	
		BASIC FREE RAM
36864	9000	
		BASIC TEXT SCREEN
35840	8C00	
		BASIC FREE RAM
25600	6400	
		COLOR MEMORY FOR HIRES SCREEN II
24576	6000	
		HIRES SCREEN II (HGR2 MODE) OR BASIC PROGRAM AND DATA STORAGE
16384	4000	
		BASIC FREE RAM FOR BASIC PROGRAM AND DATA STORAGE
2048	800	
		NOT USED
1024	400	
		RESERVED RAM FOR BASIC
00	00	

FIGURE 2 - C-64 MEMORY MAP FOR EMULATION MODE WITH CHARACTER SET

```
MEMORY ADDRESS
DECIMAL HEX
65535   FFFF  ----------------------------------------------
              8K KERNAL ROM AND HIRES SCREEN I (UNDER ROM)
57344   E000  ----------------------------------------------
                          4K I/O AREA
53248   D000  ----------------------------------------------
                 COLOR MEMORY FOR HIRES SCREEN I
52224   CC00  ----------------------------------------------
                 EMULATION PROGRAM AREA (PART B)
49152   C000  ----------------------------------------------
                          8K BASIC ROM
40960   A000  ----------------------------------------------
                 EMULATION PROGRAM AREA (PART A)
37120   9100  ----------------------------------------------
                         BASIC FREE RAM
               FOR BASIC PROGRAM AND DATA STORAGE
25600   6400  ----------------------------------------------
                COLOR MEMORY FOR HIRES SCREEN II
               OR BASIC PROGRAM AND DATA STORAGE
24576   6000  ----------------------------------------------
                  HIRES SCREEN II (HGR2 MODE)
               OR BASIC PROGRAM AND DATA STORAGE
16384   4000  ----------------------------------------------
                         BASIC FREE RAM
               FOR BASIC PROGRAM AND DATA STORAGE
 2048    800  ----------------------------------------------
                          TEXT SCREEN
 1024    400  ----------------------------------------------
                     RESERVED RAM FOR BASIC
   00     00  ----------------------------------------------
```

FIGURE 8 - C-64 MEMORY MAP FOR EMULATION OPTION ONLY

APPENDIX B

APPLE II TRANSFER PROGRAM LISTINGS

```
10   REM APPLE PROGRAM FILE NAME IS 'HELLO'
20   REM  'APPLE BASIC DRIVER ROUTINE
40   REM  '01/12/86
60   SPEED= 255
80   IF ZZ = 1 THEN 500: REM  SKIP ACKNOWLEDGEMENTS
100  REM  SET SKIP TITLE FLAG
120  ZZ = 1
140  HOME : VTAB 4
160  HTAB 14
180  REM  DISPLAY ACKNOWLEDGEMENTS AND TITLE
200  FLASH : PRINT "TRANSVERSION";: NORMAL : PRINT
220  PRINT
240  HTAB 9
260  PRINT "THE APPLE TO COMMODORE"
280  PRINT
300  HTAB 7
320  PRINT "TRANSFER/CONVERSION SYSTEM"
340  VTAB 12
360  HTAB 14
380  PRINT "COPYRIGHT 1986": PRINT
400  HTAB 20
420  PRINT "BY": PRINT
440  PRINT "  LONALD L. FINK AND THOMAS G. CLEAVER": PRINT
460  FOR L = 1 TO 5000: NEXT
480  HOME : PRINT
500  PRINT "APPLE II DRIVER PROGRAM NOW INSTALLED"
520  PRINT : PRINT
540  PRINT "START COMMODORE PROGRAM"
560  HTAB 22
580  PRINT "BEFORE PROCEEDING!!"
600  PRINT
620  PRINT "PRESS ANY KEY TO CONTINUE"
640  REM   CLEAR ERROR FLAG
660  REM  DISABLE TRANSMISSION AND RECEIVING
680  POKE 222,0: POKE 49243,255
700  POKE  - 16368,0
720  GET A$
740  POKE  - 16368,0
760  REM  CLEAR MEMORY FOR BINARY FILES
780  REM  MAXIMUM LENGTH IS 28000 BYTES
800  HIMEM: 9990
820  CLEAR
840  D$ =  CHR$ (4): REM  CTRL-D
860  POKE 222,0: POKE 49243,255
880  PRINT D$;"NOMONICO"
900  HOME
920  REM  GET FILENAME FOR TRANSFER
940  PRINT
960  PRINT "ENTER NAME OF FILE TO BE TRANSFERED"
```

```
980  PRINT
1000  PRINT "FILENAME MUST BE 16 CHARACTERS OR LESS"
1020  PRINT
1040  PRINT "THEN PRESS THE <RETURN> KEY"
1060  PRINT
1080  PRINT "PRESS '1' THEN <RETURN> KEY TO EXIT"
1100  HTAB 34: PRINT "PROGRAM": PRINT
1120  PRINT "PRESS '2' THEN <RETURN> KEY FOR"
1140  HTAB 23: PRINT "DISKETTE DIRECTORY"
1160  PRINT
1180  INPUT A$
1200  V =  VAL (A$)
1220  REM  IF V=1 EXIT PROGRAM
1240  REM  IF V=2 THEN DISPLAY DIRECTORY
1260  ON V GOTO 5780,1300
1280  GOTO 1340
1300  PRINT D$;"CATALOG"
1320  FOR L = 0 TO 4000: NEXT : GOTO 900
1340  IF  LEN (A$) < 1 OR  LEN (A$) > 16 THEN 900
1360  HOME
1380  REM  GET THE TYPE OF FILE TO BE TRANSFERED
1400  PRINT "THE FILE TO BE TRANSFERED IS"
1420  PRINT : FLASH : PRINT A$: NORMAL : PRINT : PRINT
1440  PRINT "PLEASE IDENTIFY THE TYPE OF FILE"
1460  PRINT
1480  PRINT "THAT ";: FLASH : PRINT A$;
      : NORMAL : PRINT "          IS:"
1500  PRINT
1520  PRINT "PLEASE MAKE SELECTION BY NUMBER."
1540  PRINT
1560  PRINT "PRESS '1' FOR BASIC PROGRAM"
1580  PRINT
1600  PRINT "PRESS '2' FOR BINARY FILE"
1620  PRINT
1640  PRINT "PRESS '3' FOR TEXT FILE"
1660  PRINT
1680  PRINT "PRESS '4' TO SELECT ANOTHER FILE"
1700  PRINT
1720  PRINT "PRESS '5' TO EXIT THIS PROGRAM"
1740  PRINT
1760  PRINT "PRESS '6' FOR DIRECTORY"
1780  REM  GET SELECTION AND CLEAR KBD STROBE
1800  PRINT : GET X$:V =  VAL (X$): POKE  - 16368,0
1820  ON V GOTO 2080,3180,3740,900,5780,1880
1840  REM  INVALID SELECTION TRY AGAIN
1860  GOTO 1360
1880  HOME
1900  REM  DISPLAY DIRECTORY
1920  PRINT
```

```
1940  PRINT  CHR$ (4);"CATALOG"
1960  FOR L = 1 TO 4000: NEXT
1980  REM  GO GET NEW SELECTION
2000  GOTO 1360
2020  REM  BASIC PROGRAM SELECTED
2040  REM  LOAD BASIC FILE TRANSFER ROUTINE
2060  REM  STORE NAME OF FILE IN MEMORY
2080  HOME :NAME = 970:N$ = "BASIC"
2100  PRINT "LOADING BASIC TRANSFER PROGRAM"
2120  PRINT D$;"BLOAD MLBASICTRANSFER"
2140  REM  SAVE NAME OF BASIC FILE
2160  REM  CLEAR MEMORY FOR BASIC PROGRAM
2180  GOSUB 4640: HIMEM: 38400
2200  HOME
2220  REM  MAKE EXEC FILE TO CONTROL THE ACTION
2240  PRINT "MAKING EXEC FILE NAMED 'TRANSFER BASIC'"
2260  REM  CHR$(34) IS A DOUBLE QUOTE
2280  C$ =  CHR$ (34)
2300  REM  EXEC FILE IS NAMED TRANSFER BASIC
2320  PRINT D$;"OPEN TRANSFER BASIC"
2340  PRINT D$;"DELETE TRANSFER BASIC"
2360  PRINT D$;"OPEN TRANSFER BASIC"
2380  PRINT D$;"WRITE TRANSFER BASIC"
2400  REM  ALL OUTPUT IS TO EXEC FILE
2420  REM  CREATING THE NECESSARY COMMANDS
2440  REM  INHIBIT DISK DISPLAY
2460  PRINT "NOMONICO"
2480  REM  TELL OPERATOR WHAT IS HAPPENING
2500  REM  A$ IS NAME OF FILE
2510  PRINT "NEW"
2520  PRINT "?" + C$ + "LOADING BASIC FILE "+ C$ + ";A$"
2540  PRINT "LOAD ";A$
2560  REM  LOAD BASIC PROGRAM INTO MEMORY
2620  REM TELL OPERATOR TRANSFER STARTED
2640  PRINT "?" + C$ +"TRANSFERING BASIC FILE " + A$ + C$
2660  REM  JUMP TO TRANSFER ROUTINE
2680  PRINT "CALL 768"
2700  REM   TRANSFER COMPLETED
2720  REM  TELL OPERATOR RELOADING TRANSFER PROGRAM
2740  PRINT "?" + C$+"LOADING MASTER TRANSFER PROGRAM"+ C$
2760  REM  LOAD MASTER TRANSFER PROGRAM
2780  PRINT "LOAD MASTER TRANSFER"
2800  REM  JUMP TO TRANSFER COMPLETE MESSAGE
2820  REM  SAVE NAME OF FILE AND TYPE OF FILE
2840  PRINT "GOTO 5040": PRINT "BASIC": PRINT A$
2860  REM  CLOSE EXEC FILE
2880  PRINT D$;"CLOSE TRANSFER BASIC"
2900  REM  EXEC FILE IS COMPLETED
2920  HOME
```

```
2940  REM  SAVE DRIVER PROGRAM ON DISK
2960  PRINT "SAVING MASTER TRANSFER PROGRAM"
2980  PRINT D$;"SAVE MASTER TRANSFER"
3000  HOME
3020  REM  TRANSFER BASIC FILE BY
3040  REM  EXECUTING THE NEWLY MADE EXEC FILE
3060  REM  TELL OPERATOR WHAT IS HAPPENING
3080  PRINT "EXECUTING TRANSFER BASIC FILE"
3100  PRINT
3120  PRINT  CHR$ (4);"EXEC TRANSFER BASIC"
3140  END
3160  REM  BINARY PROGRAM WAS SELECTED
3180  HOME :N$ = "BINARY"
3200  REM  TELL OPERATOR WHAT IS GOING ON
3220  PRINT "LOADING BINARY FILE TRANSFER PROGRAM"
3240  REM  LOAD BINARY TRANSFER ROUTINE
3260  PRINT  CHR$ (4);"BLOAD MLBINTRANSFER"
3280  REM  SAVE BINAY FILENAME IN MEMORY
3300  NAME = 11466: GOSUB 4640
3320  HOME
3340  PRINT "LOADING BINARY FILE NAMED "
3360  FLASH : PRINT A$;: NORMAL : PRINT
3380  REM  LOAD BINARY FILE AT SELECTED MEMORY LOCATION
3400  PRINT D$;"BLOAD";A$; + ",A$2CF0"
3420  HOME
3440  PRINT "TRANSFERING BINARY FILE"
3460  REM  JUMP TO BINARY TRANSFER ROUTINE
3480  CALL 10000
3500  REM  CHECK IF DISK ERROR OCCURED
3520  IF  PEEK (10951) <  > 0 THEN  GOTO 3560
3540  GOTO 3580
3560  PRINT : PRINT : PRINT " I/O ERROR": GOTO 5800
3580  REM  CHECK IF FILE NOT FOUND
3600  IF  PEEK (10952) <  > 0 THEN 5800
3620  REM  EVERYTHING OKAY TRANSFER COMPLETED
3640  GOTO 5160
3660  REM   ERROR OCCURED-- GET REST OF COMMANDS FROM EXEC
      FILE
3680  REM  THEN ABORT PROGRAM
3700  INPUT X$,X$: SPEED= 25:PRINT "ABORTING PROGRAM": END
3720  REM  TEXT FILE TRANSFER SECTION
3740  HOME :N$ = "TEXT"
3760  PRINT "LOADING TEXT FILE TRANSFER PROGRAM"
3780  REM  LOAD TEXT TRANSFER ROUTINE
3800  PRINT  CHR$ (4);"BLOAD MLTEXTTRANSFER"
3820  REM  SAVE FILE NAME IN SELECTED MEMORY LOCATION
3840  NAME = 11641: GOSUB 4640
3860  REM  GET TYPE OF TEXT FILE
3880  HOME : PRINT "PLEASE IDENTIFY THE TYPE TEXT FILE"
```

```
3900  PRINT "THAT ";: FLASH : PRINT A$;: NORMAL : PRINT "
      IS:"
3920  PRINT
3940  PRINT "PRESS 'R' FOR RANDOM ACCESS"
3960  PRINT
3980  PRINT "PRESS 'S' FOR SEQUENTIAL."
4000  PRINT : PRINT "PRESS 'E' FOR EXIT"
4020  GET X$
4040  REM  CLEAR KBD STROBE
4060  POKE  - 16368,0
4080  IF  LEFT$ (X$,1) = "E" THEN 5780
4100  IF  LEFT$ (X$,1) = "R" THEN 4200
4120  IF  LEFT$ (X$,1) = "S" THEN 4340
4140  REM  WRONG ANSWER TRY AGAIN
4160  GOTO 3880
4180  REM  RANDOM ACCESS FILE --- GET RECORD SIZE
4200  HOME : PRINT "ENTER SIZE OF EACH RECORD!!!"
4220  PRINT : PRINT "VALID RANGE IS '1-254' !!!"
4240  PRINT
4260  PRINT "FOLLOWED BY 'RETURN' KEY."
4280  INPUT A: IF A < 1 OR A > 254 THEN 4200
4300  REM  SAVE RECORD SIZE IN MEMORY
4320  POKE 11129,A
4340  HOME
4360  PRINT "TRANSFERING TEXT FILE "
4380  FLASH : PRINT A$;: NORMAL : PRINT
4400  REM  JUMP TO TRANSFER ROUTINE
4420  PRINT : PRINT : CALL 10000
4440  REM  CHECK FOR I/O ERRORS
4460  IF  PEEK (10944) <  > 0 THEN  GOTO 4500
4480  GOTO 4520
4500  PRINT : PRINT "I/O ERROR": GOTO 5800
4520  REM  CHECK IF NAME NOT FOUND
4540  IF  PEEK (10943) <  > 0 THEN 5800
4560  REM EVERYTHING OKAY -- DO IT AGAIN ?
4580  GOTO 5160
4600  REM  STORE NAME SUBROUTINE
4620  REM  GET PROPER DISKETTE
4640  FOR A = 1 TO  LEN (A$)
4660  POKE NAME + A, ASC ( MID$ (A$,A,1)) + 128
4680  NEXT A
4700  REM  TERMINATE NAME WITH A ZERO
4720  POKE NAME + A,0: HOME
4740  REM  GET PROPER DISKETTE
4760  REM  IF A BASIC TRANSFER --NOT WRITE PROTECTED
4780  PRINT "INSERT APPLE DISKETTE": PRINT
4800  PRINT "CONTAINING THE FILE NAMED.": PRINT
4820  FLASH : PRINT A$: PRINT : NORMAL
4840  IF N$ = "BASIC" THEN  GOTO 4880
```

```
4860  GOTO 4940
4880  PRINT "MAKE SURE DISKETTE IS ";
4900  FLASH : PRINT "NOT";: NORMAL : PRINT : HTAB 20
4920  PRINT "WRITE PROTECTED."
4940  PRINT
4960  PRINT "PRESS <RETURN> WHEN READY."
4980  PRINT
5000  INPUT X$: RETURN
5020  REM  GET NAME AND TYPE OF FILE FROM EXEC FILE
5040  INPUT N$: INPUT A$
5060  SPEED= 255
5080  REM  CHECK IF FILE TRANSFERED PROPERLY
5100  IF  PEEK (962) = 8 THEN 5160
5120  HOME : PRINT "TRANSFER FAILED": GOTO 5800
5140  REM  FILE TRANSFER COMPLETED OKAY
5160  HOME : PRINT "TRANSFER COMPLETED": PRINT
5180  PRINT " ON ";: FLASH : PRINT N$;: NORMAL
5200  PRINT " FILE NAMED ";: FLASH : PRINT A$: NORMAL
5220  PRINT
5240  PRINT "PRESS <RETURN> TO CONTINUE"
5260  PRINT
5280  INPUT X$
5300  IF N$ <  > "BASIC" THEN 5500
5320  REM  BASIC FILE WAS TRANSFERED
5340  REM  GET TRANSVERSION SOURCE DISKETTE
5360  HOME
5380  PRINT "INSERT MASTER TRANSFER DISKETTE"
5400  PRINT
5420  PRINT "PRESS ANY KEY TO CONTINUE"
5440  POKE  - 16368,0
5460  FOR A = 0 TO 127:A =  PEEK ( - 16384): NEXT A
5480  POKE  - 16368,0
5500  HOME
5520  REM DO TRANSFER AGAIN ?
5560  PRINT
5580  PRINT "PRESS 'E' TO EXIT THE PROGRAM."
5600  PRINT
5620  PRINT "PRESS <RETURN> FOR ANOTHER FILE TRANSFER."
5640  PRINT
5660  POKE  - 16368,0
5680  FOR A = 0 TO 127:A =  PEEK ( - 16384): NEXT A
5700  POKE  - 16368,0
5720  REM  'E'= 198
5740  IF A <  > 198 THEN 800
5760  REM  RESET RAM MEMORY POINTER
5780  HIMEM: 38400
5800  PRINT : PRINT "PROGRAM EXITED": END
```

```
;THE PROGRAM FILE NAME IS 'BASIC TRANSFER'
; APPLE BASIC TRANSFER PROGRAM
BASTRT EPZ !103
BASEND EPZ !175
AN1ZRO EQU !49242
AN1ONE EQU !49243
PB2    EQU !49250
       ORG $300             ; LOCATE PROGRAM AT $300
       OBJ $800
BEGIN
       LDA #21              ; SET DELAY COUNTER
       STA AMOUNT
       LDY #0
       STY DNEFLG           ; CLEAR TRANSFER STATUS FLAG
       LDA BASEND           ; CHECK IF PROGRAM IS THERE
       CMP #4
       BNE START            ; OKAY BRANCH
       LDA BASEND+1         ; CHECK HI BYTE
       CMP #8               ;
       BNE START            ; OKAY DO TRANSFER
       LDA #0               ; ABORT TRANSFER !
       STA DNEFLG           ; SET STATUS FLAG-TO NO TRANSFER
       RTS                  ; RETURN TO EXEC FILE
START
       STY FLAG             ; CLEAR END OF PROGRAM FLAG
       STA AN1ONE           ; DISABLE TRANSMISSION SET DATA
TO
                            ; ONE
NAMSND LDA FILNAM,Y         ; SEND FILENAME
       AND #$7F             ; REMOVE MSB
       PHA                  ; SAVE CHARACTER
       JSR SEND             ; SEND TO C-64
       INY                  ; MOVE READ POINTER
       PLA                  ; RETREIVE CHAR
       BNE NAMSND           ; NAME ENDS WITH ZERO
       TAY                  ; ZERO POINTER
       LDA #'B'             ; SEND FILE TYPE
       JSR SEND             ; SEND TO C-64
LOOP   LDA (BASTRT),Y       ; GET BASIC CHARACTER
       PHA                  ; SAVE CHAR
       JSR SEND             ; SEND TO C-64
       PLA                  ; RETREIVE CHAR
       BNE LOOPA            ;LOOK FOR THREE ZERO TO DETERMINE
                            ; END OF PROGRAM
       INC FLAG             ; ZERO COUNTER
       LDA FLAG             ; THREE ZEROS YET ?
       CMP #3
       BNE LOOPB            ; NO DO AGAIN
       BEQ DONE             ; YES DONE
```

```
LOOPA   LDA #0              ; CLEAR ZERO FLAG
        STA FLAG
LOOPB   JSR MOVE            ; MOVE READ POINTER
        JMP LOOP            ; GET NEXT CHARACTER
DONE    LDA #1              ; RESTORE BASIC PROGRAM START
                            ; POINTER
        STA BASTRT
        LDA #8
        STA BASTRT+1
        STA DNEFLG          ; SET DONE FLAG
        RTS
MOVE    INC BASTRT          ; INCREMENT BY ONE
        BNE MOVRTN
        INC BASTRT+1
MOVRTN RTS                  ; RETURN
SEND    SEI                 ; SEND CHAR ROUTINE
        STY YTMP            ; DISABLE INTERRUPTS PERSERVE
                            ; REGISTER
        NOP
        NOP
        NOP
SENDA   LDY PB2             ; WAIT FOR HANDSHAKE
        BPL SENDA           ; FROM C-64
        JSR DELAY           ; DELAY ONE BIT DELAY
TIMSTR STA AN1ZRO           ; SEND START BIT A LOW
        BIT $0D             ; TIMING
        BIT $0D
        BIT $0D
        JSR DELAY           ; DELAY ONE BIT
        LDY #8              ; SET COUNTER
AGAIN
        LSR                 ; TO 8 DATA BITS
        BCS ONE             ; CARRY DETERMINES WHAT STATE BIT
                            ; IS IN: LSB GOES FIRST
        BIT $0D             ; TIMING
        STA AN1ZRO          ; SEND ZERO
        BIT $0D             ; TIMING
        JMP CHECK           ; DELAY CHECK IF DONE
ONE     NOP                 ; TIMING
        STA AN1ONE          ; SEND HIGH BIT
        NOP                 ; TIMING
        NOP
        NOP
CHECK   JSR DELAY           ; DELAY ONE BIT
        DEY                 ; DECREASE CHAR COUNTER
        BNE AGAIN           ; NOT DONE BRANCH
        CMP $0D,X           ; TIMING
        CMP $0D,X
        STA AN1ONE          ; SEND  HIGH BIT
```

```
       JSR DELAY        ; TWO STOP BITS
       JSR DELAY        ; DELAY
       CLI              ; ENABLE INTERRUPTS
       LDY YTMP         ; RESTORE REGISTER
       RTS              ; RETURN
YTMP   HEX 00
FLAG   HEX 00           ; DELAYS 9*AMOUNT +15 CYCLES
DELAY:
       JSR DELAYA       ; DELAY TWICE
DELAYA:
       STX XTMP         ; PERSERVE REG
       LDX #0
DLY    INX              ; USED AS COUNTER
       CPX AMOUNT       ; DONE YET ?
       BNE DLY          ; NO DO AGAIN
       LDX XTMP         ; YES REG RESTORE
       RTS              ; RETURN
DNEFLG HEX 00
XTMP   HEX 00
AMOUNT HEX 20
FILNAM DFS $20
DLYA   END
```

```
; THE APPLE PROGRAM FILE NAME IS ' BINTRANSFER'
; APPLE BINARY TRANSFER PROGRAM!!!!
; GETS START ADDRESS
; AND LENGTH FROM DISK
; SENDS
; NAME AND START ADDRESS
; AND LENGTH
; THEN THE BODY OF THE PROGRAM
       ORG !10000          ; START LOCATION
       OBJ $800
       PLA                 ; GET AND SAVE RETURN ADDRESS
       STA RTNSVE
       PLA
       STA RTNSVE+1
       TSX                 ; GET AND SAVE STACK POINTER
       STX STACK
       PHA                 ; RESTORE RETURN ADDRESS
       LDA RTNSVE
       PHA
       LDA ZPAG            ; SAVE ZPAG LOCATIONS
       STA ZPAGSV
       LDA ZPAG+1
       STA ZPAGSV+1
       LDA SOURCE
       STA SOURVE
       LDA SOURCE+1
       STA SOURVE+1
       LDA DEST
       STA DESTVE
       LDA DEST+1
       STA DESTVE+1
AN1ZRO EQU !49242          ; I/O LOCATION FOR SENDING A ZERO
                           ; TO C-64
AN1ONE EQU !49243          ; I/O LOCATION FOR SENDINNG A ONE
                           ; TO C-64
PB2    EQU !49250
       LDA #21             ; SET BAUD RATE TO 21 LOOPS
       STA AMOUNT
       LDY #0              ; ZERO FLAGS AND DISABLE
                           ; TRANSMISSION
                           ; SEND STOP BIT A HIGH
       STY FLAG
       STA AN1ONE
       STY ERRFLG
       LDY #$FF            ; MOVE NAME AND GET LENGTH
TEXTC
       INY
       LDA FILNAM,Y
       STA TEXTB,Y         ; NAME ENDS WITH ZERO
```

```
       BNE TEXTC
       STY LENGTH          ; SAVE LENGTH
       INY
       LDA #$60            ; FOLLOW NAME WITH A RTS OPCODE
       STA TEXTB,Y
       JSR START           ; GET FILE NAME IN DIRECTORY
                           ; GET LOCATION OF T/S SECTOR
       LDA ERRFLG          ; ERROR OCCURED PRINT
       BNE TEXTE           ; YES BRANCH
       LDA FNDFLG          ; FOUND FILENAMEPRINT
       BEQ TEXTF           ; YES BRANCH
       JSR MSG             ; DISPLAY ERROR MESSAGE
       HEX 8D
       ASC "BINARY FILE NOT FOUND"
       HEX 8D00
       JSR DISNAM          ; DISPLAY NAME
TEXTE
       RTS                 ; RETURN
RTNSVE HEX 0000
STACK  HEX 00
ZPAGSV HEX 0000
SOURVE HEX 0000
DESTVE HEX 0000
STRLOC HEX 0000
LENLOC HEX 0000
TEXTF
       JSR MSG             ; DISPLAY MESSAGE
       HEX 8D
       ASC "BINARY FILE FOUND"
       HEX 00
       JSR DISNAM          ; DISPLAY FILENAME
ONETSL
       LDA #$1             ; SET FOR READ COMMAND
       STA CMD
       LDA #BUFFER         ; POINT TO DATA BUFFER
       STA BUF
       LDA /BUFFER
       STA BUF+$1
       LDA /IOB            ; POINT TO IOB BLOCK
       LDY #IOB
       JSR RWTS            ; READ SECTOR
       LDA ERRFLG          ; ERROR OCCURED PRINT
       BNE TEXTE           ; YES BRANCH
       LDY #$C             ; GET TRACK NUMBER OF DATA SECTOR
       LDA BUFFER,Y
       STA TRACK
       INY
       LDA BUFFER,Y        ; GET SECTOR NUMBER OF DATA
SECTOR
```

```
        STA SECTOR
        ORA TRACK           ; IF BOTH ZERO END
        BEQ TEXTE           ; YES THEN END
        LDA 01              ; SET FOR READ COMMAND
        STA CMD
        LDA /IOB            ; POINT TO IOB BLOCK
        LDY #IOB
        JSR RWTS            ; READ DATA SECTOR
        LDA ERRFLG          ; ERROR OCCURED PRINT
        BNE TEXTE           ; YES BRANCH
                            ; GET THE DATA
                            ; OUT OF FIRST SECTOR ONLY
        LDY #0
        LDA BUFFER,Y        ; GET START ADDRESS
        STA STRLOC
        STA ADDR
        INY
        LDA BUFFER,Y
        STA STRLOC+1
        STA ADDR+1
        INY
        LDA BUFFER,Y        ; GET LENGTH
        STA LENLOC
        INY
        LDA BUFFER,Y
        STA LENLOC+1
        JSR MSG             ; DISPLAY MESSAGE
        HEX 8D
        ASC "START ADDRESS IS : "
        HEX 8D00
        JSR CVHD            ; CONVERT START ADDRESS
                            ; TO ASCII AND DISPLAY
        JSR MSG             ; DISPLAY MESSAGE
        HEX 8D
        ASC "PROGRAM LENGTH IS : "
        HEX 8D00
        LDA LENLOC          ; SET UP FOR CONVERSION
        STA ADDR
        LDA LENLOC+1
        STA ADDR+1
        JSR CVHD            ; CONVERT TO ASCII
                            ; AND DISPLAY LENGTH
        JSR SNDNAM          ; SEND NAME TO C-64
        LDA #'M'            ; SEND FILE TYPE TO C-64 ---
BINARY
        JSR SEND
        LDA STRLOC          ; SEND START ADDRESS
        JSR SEND
        LDA STRLOC+1
```

```
        JSR SEND
        LDA LENLOC          ; SEND LENGTH
        JSR SEND
        LDA LENLOC+1
        JSR SEND
        CLC
        LDA #DLYA           ; GET START TRANSFER LOCATION
                            ; -LOW BYTE
        STA ZPAG
        ADC LENLOC          ; ADD LENGTH
        STA LENLOC          ; SAVE END TRANSFER LOCATION
                            ; -LOW BYTE
        LDA /DLYA           ; GET START TRANSFER LOCATION
                            ; -HIGH BYTE
        STA ZPAG+1
        ADC LENLOC+1
        STA LENLOC+1        ; SAVE END TRANSFER LOCATION
                            ; -HIGH BYTE
        JSR MSG             ; DISPLAY MESSAGE
        ASC "WAITING ON COMMODORE PROGRAM"
        HEX 8D
        ASC "CHECK COMMODORE PROGRAM"
        HEX 8D00
        LDA #DLYA           ; RESTORE START ADDRESS
                            ; TO ZERO PAGE POINTER
        STA ZPAG
        LDA /DLYA
        STA ZPAG+1
        LDY #0              ; START TRANSFER OF BINARY DATA
        LDA (ZPAG),Y        ; GET FIRST LOCATION
        JSR SEND            ; SEND TO C-64
        JSR INCZ            ; MOVE READ POINTER N BY ONE
        JSR HOME            ; CLEAR SCREEN
        JSR CHCK            ; CHECK IF DONE
        BEQ AGAEND          ; BRANCH IF DONE
LOOP
        LDY #0              ; GET REST OF DATA
        LDA (ZPAG),Y
        JSR SEND            ; SEND TO C-64
        JSR CHCK            ; DONE YET ?
        BEQ AGAEND          ; YES BRANCH
        JSR INCZ            ; MOVE READ POINTER
        JMP LOOP            ; DO IT AGAIN SAM
CHCK:
        LDA ZPAG            ; CHECK TO SEE IF READ POINTER
                            ; IS THE SAME AS END LOCATION
        CMP LENLOC
        BNE CHCKND
        LDA ZPAG+1
```

```
        CMP LENLOC+1        ; IF ZERO FLAG SET THEN DONE
CHCKND
        RTS
AGAEND:
        LDA ZPAGSV          ; RESTORE ZPAG LOCATIONS
        STA ZPAG
        LDA ZPAGSV+1
        STA ZPAG+1
        LDX STACK           ; RESTORE STACK POINTER
        TXS
        LDA RTNSVE+1        ; RESTORE RETURN ADDRESS
        PHA
        LDA RTNSVE
        PHA
        LDA SOURVE          ; RESTORE ZERO PAGE LOCATIONS
        STA SOURCE
        LDA SOURVE+1
        STA SOURCE+1
        LDA DESTVE
        STA DEST
        LDA DESTVE+1
        STA DEST+1
        RTS
XSAVE   HEX 00
ASAVE   HEX 00
SNDNAM
        LDY #255            ; SEND FILENAME
NAME
        INY
        LDA FILNAM,Y        ; GET CHAR
        AND #$7F            ; STRIP MSB
        JSR SEND            ; SEND TO C-64
        BNE NAME            ; ZERO ENDS NAME
        RTS                 ; RETURN
SEND
        SEI                 ; SEND CHAR TO C-64
        STY YTMP            ; DISABLE INTERRUPTS
                            ; AND PERSERVE REGISTERS
        STX XSAVE
        STA ASAVE
        NOP
        NOP
        NOP
SENDA
        LDX PB2             ; GET STATUS FROM C-64
        BPL SENDA           ; WAIT FOR HANDSHAKE
        JSR DELAY           ; DELAY ONE BIT DELAY
TIMSTR
        STA AN1ZRO          ; SEND START BIT A LOW
```

```
        BIT $0D              ; TIMING
        BIT $0D
        BIT $0D
        JSR DELAY            ; DELAY ONE BIT
        LDY #8               ; SEND 8 DATA BITS
AGAIN
        LSR                  ; USE CARRY TO DETERMINE STATE
                             ; LSB GOES FIRST
        BCS ONE              ; IF ONE BRANCH
        BIT $0D              ; TIMING
        STA AN1ZRO           ; SEND ZERO
        BIT $0D              ; TIMING
        JMP CHECK            ; DELAY AND CHECK IF DONE
ONE
        NOP                  ; TIMING
        STA AN1ONE           ; SEND A HIGH
        NOP                  ; TIMING
        NOP
        NOP
CHECK
        JSR DELAY            ; DELAY ONE BIT DELAY
        DEY                  ; UPDATE BIT COUNTER
        BNE AGAIN            ; DONE NO BRANCH
        CMP $0D,X            ; YES MORE TIMMING
        CMP $0D,X
        STA AN1ONE           ; SEND TWO STOP BITS
        JSR DELAY            ; DELAY TWO BITS
        JSR DELAY
        CLI                  ; ENABLE INTERRUPTS
                             ; AND RESTORE REGISTERS
        LDY YTMP
        LDX XSAVE
        LDA ASAVE
        RTS                  ; RETURN
YTMP    HEX 00
VALUE   HEX 00
FLAG  HEX 00                 ; DELAYS 9*AMOUNT +15 CYCLES
DELAY:                       ; DELAY BETWEEN BITS
        JSR DELAYA           ; DO ROUTINE TWICE
DELAYA:
        STX XTMP             ; PERSERVE REGISTER USED AS
COUNTER
        LDX #0
DLY
        INX
        CPX AMOUNT           ; CHECK IF DONE
        BNE DLY              ; NO THEN DO AGAIN
        LDX XTMP             ; RESTORE REG
        RTS                  ; RETURN
```

```
LENGTH HEX 00
       HEX 00
XTMP   HEX 00
AMOUNT HEX 20
DISNAM
       JSR MSG             ; DISPLAY NAME ROUTINE
       HEX 8D
TEXTB  DFS $20             ; NAME STORED HERE
       HEX 00
       RTS
; INPUT/OUTPUT CONTROL BLOCK AS
; PER APPLE COMPUTER'S INSTRUCTIONS
; PLEASE CONSULT APPLE DOS 3.2 MANUAL
; PAGES 91-98, AND 123-138.
;
;
;
IOB    HEX 01
SLOT   HEX 60              ; SLOT 6
DRIVE  HEX 01              ; DRIVE 1
VOL    HEX 00              ; ANY VOLUME
TRACK  HEX 11              ; TRACK TO BE READ/WRITTEN
SECTOR HEX 00              ; SECTOR TO BE READ/WRITTEN
DCT    ADR DEVICE          ; POINTER TO DEVICE CHAR. TABLE
BUF    ADR BUFFER          ; POINTER TO BUFFER AREA.
UNUSED HEX 0000
CMD    HEX 00              ; COMMAND CODE GOES HERE.
ERROR  HEX 00              ; ERROR CODE RETURNED HERE.
ACTVOL HEX 00              ; ACTUAL VOLUME FOUND
PRVSLT HEX 60              ; PREVIOUS SLOT
PRVDRV HEX 01              ; PREVIOUS DRIVE
;
;
; DEVICE CHARACTERISTICS TABLE
; VERBATIM ALA APPLE.
; DEVICE HEX 00
       HEX 01
       HEX EF
       HEX D8
START:                     ; FIND FILE NAME IN DIRECTORY
       LDA #$11            ; SET TRACK NUMBER
       STA TRACK
       LDA #$F             ; SET SECTOR NUMBER
                           ; TO FIRST DIRECTORY SECTOR
       STA SECTOR
       LDA 01              ; SET FOR READ COMMAND
       STA CMD
       STA FNDFLG          ; RESET FOUND NAME FLAG
DOIT:                      ; POINT I/O CONTROL BLOCK
```

```
        LDA /IOB
        LDY #IOB
        JSR RWTS            ; READ SECTOR
        LDA ERRFLG          ; ERROR OCCURED
        BNE EXIT            ; YES BRANCH
        LDY #$B             ; POINT TO FIRST FILE NAME
        STY FILNUM
        LDA #FILNAM         ; SETUP FOR COMPARSION
                            ; OF DESIRED NAME AND
                            ; UNKNOWN NAME
        STA SOURCE
        LDA /FILNAM
        STA SOURCE+$1
DOITA
        LDY FILNUM          ; SET UP FOR NAME
        JSR PRTFIL          ; FIND FILE NAME AND
                            ; TRACK AND SECTOR
                            ; OF T/S LIST SECTOR
        LDA FNDFLG          ; IF ZERO FILE FOUND
        BEQ EXIT            ; YES BRANCH
        CLC                 ; GET POINTER TO NEXT FILE NAME
        LDA FILNUM          ; GET PRESENT LOCATION
        ADC #$23
        STA FILNUM          ; SAVE NEW PRESENT
        BNE DOITA           ; GO DO IT AGAIN
;
; WHEN BOTH BYTES OF LINK ARE
; ZERO YOU ARE THROUGH.
;
        LDY #$1
        LDA BUFFER,Y        ; GET AND SAVE TRACK NUMBER
        STA TRACK
        ORA BUFFER+$1,Y     ; GET SECTOR NUMBER
        BEQ EXIT            ; IF BOTH ZERO END
        LDA BUFFER+$1,Y
        STA SECTOR          ; SAVE SECTOR
        JMP DOIT            ; TRY AGAIN
EXIT
        RTS
;
PRTFIL:
        LDA BUFFER,Y        ; GET STATUS OF FILE
        CMP #$FF            ; IS IT A DELETED FILE ?
        BEQ PRTX            ; YES BRANCH
        STA TRACK           ; SAVE TRACK NUMBER
        INY
        LDA BUFFER,Y
        STA SECTOR          ; SAVE SECTOR NUMBER
        INY
```

```
        LDA BUFFER,Y        ; CHECK TYPE OF FILE
        AND #$4             ; BINARY IS $4 OR $84
        BEQ PRTX            ; NO THEN RETURN
        INY
        JMP CHKNAM          ; YES CHECK NAME OF FILE IF
CORRECT
PRTX:
        RTS
; MSG PRINTS AN ASCII STRING TO
; THE VIDEO SCREEN.
MSG
        PLA                 ; GET READ POINTER FOR STACK
        STA ZPAG
        PLA
        STA ZPAG+$1
        JSR INCZ            ; ADVANCE READ POINTER BY ONE
        STY YSAVE           ; PERSERVE REGS
        LDY 00
LOOP2
        LDA (ZPAG),Y        ; GET CHARACTER
        BEQ LOOP3           ; ZERO ENDS MESSAGE
        JSR PUTC            ; DISPLAY TO SCREEN
        JSR INCZ            ; MOVE READ POINTER
        JMP LOOP2           ; DO AGAIN
LOOP3
        JSR INCZ            ; MOVE READ POINTER
        LDY YSAVE           ; RESTORE REGS
        JMP (ZPAG)          ; CONTINUE WITH PROGRAM
;
YSAVE   HEX 00
ZPAG    EPZ $00
PUTC    EQU $FDED
HOME    EQU $FC58
INCZ
        INC ZPAG            ; INCREMENT READ POINTER
        BNE INCZ1
        INC ZPAG+$1
INCZ1
        RTS
;
; LINK DISPLACEMENT
;
SOURCE EPZ $06
DEST    EPZ SOURCE+$2
STRCOM
        LDY #$FF            ; COMPARE TO STRINGS
STRCM1
        INY                 ; INCREMENT READ POINTER
        DEX                 ; CONTAINS LENGTH
```

```
        BEQ STRCM2          ; YES THEN END
        SEC
        LDA (SOURCE),Y      ; CHECK STRING
        SBC (DEST),Y        ; CHECK BY SUBTRACTING
        BEQ STRCM1
        RTS                 ; IF EQUAL ZERO FLAG SET
;
STRMAB
        LDA #$0             ; SET ZERO FLAG
        RTS                 ; HANDLE 256TH COMPARE.
;
STRCM2
        SEC                 ; CHECK LAST CHAR
        LDA (SOURCE),Y
        SBC (DEST),Y
        BNE STRMCA
STRMAA
        INY                 ; CHECK IF SUBSET OF STRING
        CPY #$1E            ; MAXIMUM LENGTH OF FILE NAME
        BEQ STRMAB          ; YES THEN END
        LDA (DEST),Y        ; CHECK FOR SPACES TO END
        CMP #$A0
        BEQ STRMAA
        LDA #1              ; NOT A SPACE THEN END
STRMCA
        RTS
CHKNAM:                     ; CHECK FILE NAME
                            ; AGAINST UNKNOWN FILE NAME
        CLC
        TYA                 ; UPDATE ZERO PAGE POINTER
        ADC #BUFFER
        STA DEST            ; HOLDS LOCATION
                            ; OF UNKNOWN FILE NAME
        LDA /BUFFER
        ADC #0
        STA DEST+$1
        LDX LENGTH          ; GET LENGTH OF KNOWN FILE NAME
        JSR STRCOM          ; CHECK NAMES
        STA FNDFLG          ; SAVE STATUS
        RTS
FILNUM HEX 00
FNDFLG HEX 00
ERRFLG HEX 00
RWTS:                       ; READ SECTOR
                            ; AND DISPLAY ERROR IF ANY
        JSR $BD00           ; READ SECTOR
        BCC RWTSND          ; NO ERROR BRANCH
        LDY #$D             ; GET ERROR LOCATION
                            ; AND SET ERROR FLAG
```

```
        STY ERRFLG
        LDA IOB,Y           ; GET ERROR
        CMP #$10
        BEQ WRTPRT          ; WRITE PROTECT ERROR
        CMP #$20
        BEQ VOLERR          ; VOLUME ERROR
        CMP #$40
        BEQ DRVERR          ; DRIVE ERROR
        CMP #$80
        BEQ READRR          ; READ ERROR
        JSR DSKERR          ; DISPLAY DISK ERROR
        JSR MSG             ; DISPLAY ERROR
        ASC "UNDEFINED ERROR"
        HEX 8D
        HEX 00
RWTSND
        RTS
DSKERR
        JSR MSG             ; DISPLAY DISK ERROR
        HEX 8D
        ASC "DISK ERROR OCCURED"
        HEX 8D
        HEX 00
        RTS
WRTPRT
        JSR DSKERR
        JSR MSG             ; DISPLAY ERROR
        HEX 8D
        ASC "WRITE PROTECT ERROR"
        HEX 8D
        HEX 00
        RTS
VOLERR
        JSR DSKERR
        JSR MSG             ; DISPLAY ERROR
        HEX 8D
        ASC "VOLUME ERROR"
        HEX 8D
        HEX 00
        RTS
DRVERR
        JSR DSKERR
        JSR MSG             ; DISPLAY ERROR
        HEX 8D
        ASC "DRIVE ERROR"
        HEX 8D
        HEX 00
        RTS
READRR
```

```
        JSR DSKERR
        JSR MSG             ; DISPLAY ERROR
        HEX 8D
        ASC "READ ERROR"
        HEX 8D
        HEX 00
        RTS
; :*****************
; :
; :
; : HEX TO DECIMAL
; : CONVERSION AND DISPLAY
;
CVHD:
        PHA                 ; PRESERVE REGS
        TXA
        PHA
        LDX 4               ; MAXIMUM LENGTH
        STX LEAD0
PINT1:
        LDA #ZERO
        STA DIGIT
PINT2:
        LDA ADDR            ; GET LOW BYTE
        CMP T10L,X          ; SET CARRY
        LDA ADDR+$1         ; GET HIGH BYTE
        SBC T10H,X
        BLT PINT3
        STA ADDR+$1         ; SAVE HIGH BYTE
        LDA ADDR            ; GET LOW BYTE
        SBC T10L,X
        STA ADDR            ; SAVE LOW BYTE
        INC DIGIT
        JMP PINT2
PINT3
        LDA DIGIT
        CPX #$0
        BEQ PINT5
        CMP #ZERO
        BEQ PINT4
        STA LEAD0
PINT4
        BIT LEAD0
        BPL PINT6
PINT5
        JSR PUTC            ; DISPLAY VALUE
PINT6
        DEX
        BPL PINT1
```

```
        PLA                    ; RESTORE REGISTERS
        TAX
        PLA
        RTS
;
;
T10L    HEX 010A64E810
T10H    HEX 0000000327
;
DIGIT   HEX 00
LEAD0   HEX 00
ADDR    HEX 0000
ZERO    EQU $B0
BUFFER  DFS $100
FILNAM  DFS $20
        HEX 00
FILNA:
        ORG $2CF0
DLYA    EQU *
        END
```

```
; APPLE TEXT TRANSFER PROGRAM
; GETS NAME AND TEXT
; FROM DISK A SECTOR A TIME
       ORG !10000         ; START LOCATION
       OBJ $800
       PLA                ; PULL RETURN ADDRESS AND SAVE
       STA RTNSVE
       PLA
       STA RTNSVE+1
       TSX                ; SAVE STACK POINTER
       STX STACK
       PHA                ; RESTORE RETURN ADDRESS
       LDA RTNSVE
       PHA
       LDA ZPAG           ; SAVE ZERO PAGE LOCATIONS
       STA ZPAGSV
       LDA ZPAG+1
       STA ZPAGSV+1
       LDA SOURCE
       STA SOURVE
       LDA SOURCE+1
       STA SOURVE+1
       LDA DEST
       STA DESTVE
       LDA DEST+1
       STA DESTVE+1
AN1ZRO EQU !49242        ;LOCATION TO OUTPUT A ZERO TO C-64
AN1ONE EQU !49243        ; LOCATION TO OUTPUT A ONE TO C-64
PB2    EQU !49250         ; LOCATION TO INPUT
                          ; HANDSHAKE FROM C-64
       LDA #21            ; SET BAUD RATE COUNTER
       STA AMOUNT
       LDY #0             ; DISABLE TRANSMISSION BY
                          ; SENDING A STOP BIT
       STY FLAG
       STA AN1ONE
       STY ERRFLG         ; RESET ERROR FLAG
       STY RECCNT+1      ; SET RECORD NUMBER COUNTER TO ONE
                          ; -HIGH BYTE TO ZERO
       STY BEGREC         ; SET BEGIN RECORD FLAG TO ZERO
       INY                ; SET RECORD POSITION
                          ; TO FIRST BYTE - TO ONE
       STY RECCNT         ; RECORD NUMBER
       STY RECPOS         ; RECORD POSITION POINTER
       LDY #$FF           ; MOVE FILENAME
TEXTC
       INY
       LDA FILNAM,Y       ; GET CHAR
       STA TEXTB,Y        ; SAVE CHAR
```

```
       BNE TEXTC          ; ZERO ENDS NAME
       STY LENGTH         ; SAVE LENGTH
       INY
       LDA #$60           ; END NAME WITH A RTS OPCODE
       STA TEXTB,Y
       JSR START          ; FIND FILE AND CHECK TYPE
       LDA ERRFLG         ; NON-ZERO THEN I/O ERROR
       BNE TEXTE          ; ZERO EVERYTHING OKAY
       LDA FNDFLG         ; NON-ZERO NAME NOT FOUND
       BEQ TEXTF          ; ZERO NAME FOUND
       JSR MSG            ; DISPLAY ERROR MESSAGE
       HEX 8D
       ASC "TEXT FILE NOT FOUND"
       HEX 8D00
       JSR DISNAM         ; DISPLAY NAME
TEXTE
       RTS                ; RETURN
RTNSVE HEX 0000
STACK  HEX 00
ZPAGSV HEX 0000
SOURVE HEX 0000
DESTVE HEX 0000
TEXTF
       JSR MSG            ; DISPLAY MESSAGE
       HEX 8D
       ASC "TEXT FILE FOUND"
       HEX 00
       JSR DISNAM         ; DISPLAY NAME
       JSR SNDNAM         ; SEND NAME TO C-64
       JMP ONETSL         ; GET FILE AND SEND
AGBEND
       JMP AGAEND         ; GOTO END
NEWTSL:                 ; GET NEXT T/S LIST SECTOR LOCATION
       LDY #1
       LDA BUFFTS,Y       ; GET TRACK
       ORA BUFFTS+$1,Y    ; GET SECTOR
       BEQ AGBEND         ; IF BOTH ZERO THEN END
       LDA BUFFTS,Y
       STA TRACK          ; SAVE TRACK NUMBER
       LDA BUFFTS+$1,Y    ; SAVE SECTOR NUMBER
       STA SECTOR
ONETSL
       LDA #$1            ; SET FOR READ
       STA CMD
       LDA #BUFFTS        ; POINT TO TRACK/SECTOR BUFFER
       STA BUF
       LDA /BUFFTS
       STA BUF+$1
       LDA /IOB           ; POINT TO IOB BLOCK
```

```
        LDY #IOB
        JSR RWTS            ; READ SECTOR
                            ; AND DISPLAY ERROR IF ANY
        LDA ERRFLG          ; CHECK IF ERRORS
        BNE AGAEND          ; ZERO OKAY
        LDY #$0B            ; SET POINTER TO FIRST T/S PAIR
        STY SCTBYT          ; HOLD PREVIOUS SECTOR
AGAN
        LDY SCTBYT          ; GET SECTOR PAIR NUMBER
        CPY #$FF            ; ANY MORE ?
        BEQ NEWTSL          ; NO GET NEW T/S SECTOR
        INY
        LDA BUFFTS,Y        ; YES GET TRACK NUMBER
        STA TRACK
        INY
        STY SCTBYT          ; UPDATE POINTER
        LDA BUFFTS,Y        ; SAVE SECTOR NUMBER
        STA SECTOR
        ORA TRACK           ; IF BOTH ZERO THEN END
                            ; OR UPDATE RECORD
        BEQ AGANA
        LDA 01              ; READ COMMAND
        STA CMD
        LDA #BUFFER         ; POINT TO DATA BUFFER
        STA BUF
        LDA /BUFFER
        STA BUF+$1
        LDA /IOB            ; POINT TO IOB LOC
        LDY #IOB
        JSR RWTS            ; READ SECTOR TO BUFFER
        LDA ERRFLG          ; ZERO IS OKAY
        BNE AGAEND          ; ERROR BRANCH
                            ; GET THE DATA
        JSR NDSCT           ; SEND SECTOR TO C-64
        JMP AGAN            ; GET NEXT SECTOR
                            ; AND DO IT AGAIN SAM
AGANA
        LDA RNDFLG          ; CHECK FILE TYPE
        BEQ AGAEND          ; IF SEQ THEN BRANCH
        JSR RECNUM          ; UPDATE BY ONE SECTOR
        JMP AGAN            ; DO AGAIN
AGAEND:                     ; RESTORE ZERO PAGE LOCATIONS
        LDA RNDFLG          ; CHECK FILE TYPE
        BEQ AGAENN          ; SKIP IF SEQ
        LDA #0              ; SEND FOUR ZERO'S
        JSR SEND            ; TO END TRANSFER
        JSR SEND
        JSR SEND
        JSR SEND
```

```
AGAENN:
        LDA ZPAGSV
        STA ZPAG
        LDA ZPAGSV+1
        STA ZPAG+1
        LDX STACK           ; RESTORE STACK POINTER
        TXS
        LDA RTNSVE+1        ; RESTORE RETURN ADDRESS
        PHA
        LDA RTNSVE
        PHA
        LDA SOURVE          ; RESTORE ZERO PAGE LOCATIONS
        STA SOURCE
        LDA SOURVE+1
        STA SOURCE+1
        LDA DESTVE
        STA DEST
        LDA DESTVE+1
        STA DEST+1
        RTS
RECCNT  HEX 00              ; HOLDS RECORD
NUMBER  HEX 00
RECPOS  HEX 00              ; HOLD POSITION IN RECORD
RECNUM                      ; UPDATE RECORD NUMBER
                            ; AND POSITION BY ONE SECTOR
        LDX #0              ; ZERO POSITION POINTER
RECNMA
        JSR MVEREC          ; MOVE POSITION BY ONE
                            ; UPDATE RECORD IF NEEDED
        INX                 ; MOVE POINTER TO NEXT POSITION
                            ; ARE WE DONE ?
        BNE RECNMA          ; NO THEN DO AGAIN
        RTS                 ; YES DONE
MVEREC:                     ; MOVE RECORD POSITION BY ONE
        LDY RECPOS          ; GET POSITION
        INY                 ; ADD ONE
        CPY RNDFLG          ; CHECK IF END OF RECORD
        BNE MVERCB          ; NO THEN BRANCH
        INC RECCNT          ; YES UPDATE RECORD COUNT BY ONE
        BNE MVERCA          ; WRAP AROUND OCCUR ?
        INC RECCNT+1        ; WRAP AROUND UPDATE HIGH BYTE
MVERCA
        LDY #1              ; SET RECORD POSITION
                            ; TO FIRST POSITION
MVERCB
        STY RECPOS          ; UPDATE RECORD POINTER
        RTS                 ; RETURN
NDSCT:                     ; SENDS ONE SECTOR OF DATA TO C-64
        LDA RNDFLG          ; CHECK FILE TYPE
```

```
        BNE SNDSCT          ; RANDOM ACCESS JUMP
        LDX #0              ; RESET POINTER AND SEND SEQ
SECTOR
NDSCA
        LDA BUFFER,X        ; GET DATA
        AND #$7F            ; STRIP MSB
        JSR SEND            ; SEND TO C-64
        BEQ NDSCB           ; ZERO ENDS SECTOR
        ORA #$80            ; RESTORE MSB
        JSR PUTC            ; DISPLAY
        INX                 ; MOVE READ POINTER
        BNE NDSCA           ; DO AGAIN
NDSCB
        RTS                 ; RETURN
SCTCNT HEX 00
BEGREC HEX 00
SNDSCT:                     ; SEND RANDOM ACCESS SECTOR
        LDX #0              ; ZERO SECTOR POSITION POINTER
SNDSCA
        LDA BUFFER,X        ; GET DATA
        AND #$7F            ; STRIP MSB
        BNE SNDSCD          ; VALID DATA THEN BRANCH
        LDY BEGREC          ; CHECK IF BEGINNING OF RECORD
        CPY #0
        BEQ SNDSCB          ; YES THEN BRANCH
        STA BEGREC          ; RESET FLAG
        JSR SEND            ; SEND RECORD END
SNDSCB
        JSR MVEREC          ; INCREASE RECORD POSITION BY ONE
        JMP SNDSCC          ; CONTINUE WITH SECTOR
SNDSCD:                     ; VALID DATA
        PHA                 ; SAVE DATA
        LDY BEGREC          ; BEGINNING OF RECORD ?
        BNE SNDSCE          ; NO THEN BRANCH
        LDA RECCNT          ; SEND RECORD NUMBER-- LOW BYTE
        JSR SEND
        LDA RECCNT+1        ; HIGH BYTE
        JSR SEND
        LDA RECPOS          ; SEND RECORD POSITION
        JSR SEND
        INC BEGREC          ; SET RECORD BEGIN FLAG
        BNE SNDSCE          ; BE SURE ITS SET
        INC BEGREC
SNDSCE
        JSR MVEREC          ; MOVE RECORD POSITION BY ONE
        PLA                 ; GET DATA
        JSR SEND            ; SEND TO C-64
        ORA #$80            ; SET MSB FOR DISPLAY
        JSR PUTC            ; DISPLAY DATA
```

```
SNDSCC
        INX                     ; INCREASE SECTOR POSITION
POINTER
        BNE SNDSCA              ; IF NOT DONE THEN
                                ; GET NEXT DATA BYTE
        RTS                     ; DONE RETURN
XSAVE   HEX 00
ASAVE   HEX 00
SNDNAM
        LDY #255                ; SEND NAME AND TYPE TO C-64
NAME
        INY
        LDA FILNAM,Y            ; GET CHAR
        AND #$7F                ; STRIP MSB
        STA VALUE               ; SAVE CHAR
        JSR SEND                ; SEND TO C-64
        LDA VALUE               ; RESTORE CHAR
        BNE NAME
        LDA RNDFLG              ; GET FILE TYPE
        BEQ NAMEA               ; ZERO IS SEQ
        LDA #'R'                ; RANDOM
        JSR SEND                ; SEND TO C-64
        LDA RNDFLG              ; GET RECORD LENGTH
        INC RNDFLG              ; INCREASE BY ONE THE RECORD
LENGTH
        JMP SEND                ; SEND TO C-64
NAMEA:
        LDA #'S'
SEND
        SEI                     ; DISABLE INTERRUPTS
        STY YTMP                ; PERSERVE REGS
        STX XSAVE
        STA ASAVE
        NOP
SENDA
        LDX PB2                 ; WAIT FOR C-64 HANDSHAKE
        BPL SENDA
        JSR DELAY               ; DELAY ONE BIT
TIMSTR
        STA AN1ZRO              ; SEND ONE START BIT
        BIT $0D                 ; TIMING
        BIT $0D
        BIT $0D
        JSR DELAY               ; DELAY ONE BIT
        LDY #8                  ; SEND 8 DATA BITS
AGAIN
        LSR                     ; LSB FIRST
        BCS ONE                 ; CARRY DETERMINES WHAT TO SEND
        BIT $0D                 ; TIMING
```

```
        STA AN1ZRO          ; SEND ZERO
        BIT $0D             ; TIMING
        JMP CHECK           ; DELAY AND CHECK IF DONE
ONE
        NOP                 ; TIMING
        STA AN1ONE          ; SEND ONE BIT
        NOP                 ; TIMING
        NOP
        NOP
CHECK
        JSR DELAY           ; DELAY ONE BIT
        DEY                 ; DECREASE COUNTER
        BNE AGAIN           ; NOT DONE DO AGAIN
        CMP $0D,X           ; TIMING
        CMP $0D,X
        STA AN1ONE          ; SEND TWO STOP BITS
        JSR DELAY           ; DELAY TWO BIT DELAYS
        JSR DELAY
        CLI                 ; ENABLE INTERRUPTS
        LDY YTMP            ; RESTORE REGS
        LDX XSAVE
        LDA ASAVE
        RTS                 ; RETURN
SCTBYT  HEX 00
YTMP    HEX 00
VALUE   HEX 00
FLAG    HEX 00              ; DELAYS 9*AMOUNT +15 CYCLES
DELAY:                      ; DELAYS ONE BIT
        JSR DELAYA          ; DO ROUTINE TWICE
DELAYA:
        STX XTMP            ; PRESERVE REG
        LDX #0
DLY
        INX                 ; USED AS COUNTER
        CPX AMOUNT          ; DONE YET ?
        BNE DLY             ; NO DO AGAIN
        LDX XTMP            ; RESTORE REG
        RTS                 ; RETURN
LENGTH  HEX 00
        HEX 00
XTMP    HEX 00
AMOUNT  HEX 20
DISNAM
        JSR MSG             ; DISPLAY NAME
        HEX 8D
TEXTB DFS $20
        HEX 00
        RTS
; INPUT/OUTPUT CONTROL BLOCK AS
```

```
; PER APPLE COMPUTER'S INSTRUCTIONS
; PLEASE CONSULT APPLE DOS 3.2 MANUAL
; PAGES 91-98, AND 123-138.
;
;
;
IOB     HEX 01
SLOT    HEX 60              ; SLOT 6
DRIVE   HEX 01              ; DRIVE 1
VOL     HEX 00              ; ANY VOLUME
TRACK   HEX 11              ; TRACK TO BE READ/WRITTEN
SECTOR  HEX 00              ; SECTOR TO BE READ/WRITTEN
DCT     ADR DEVICE          ; POINTER TO DEVICE CHAR. TABLE
BUF     ADR BUFFER          ; POINTER TO BUFFER AREA.
UNUSED  HEX 0000
CMD     HEX 00              ; COMMAND CODE GOES HERE.
ERROR   HEX 00              ; ERROR CODE RETURNED HERE.
ACTVOL  HEX 00              ; ACTUAL VOLUME FOUND
PRVSLT  HEX 60              ; PREVIOUS SLOT
PRVDRV  HEX 01              ; PREVIOUS DRIVE
;
;
; DEVICE CHARACTERISTICS TABLE
; VERBATIM ALA APPLE.
;
DEVICE  HEX 00
        HEX 01
        HEX EF
        HEX D8
START:                      ; FIND FILE NAME IN DIRECTORY
        LDA #$11            ; TO DIRECTORY TRACK NUMBER 17
        STA TRACK
        LDA #$F             ; TO FIRST SECTOR OF DIRECTORY
        STA SECTOR
        LDA 01              ; READ COMMAND
        STA CMD
        STA FNDFLG          ; RESET FOUND NAME FLAG
DOIT:
        LDA /IOB            ; SET POINTER TO IOB BLOCK
        LDY #IOB
        JSR RWTS            ; READ SECTOR
        LDA ERRFLG          ; ZERO OKAY NO ERROR OCCURED
        BNE EXIT            ; YES EXIT
        LDY #$B             ; POINT TO FIRST FILE NAME BYTE
        STY FILNUM
        LDA #FILNAM         ; POINT TO DESIRED FILENAME
        STA SOURCE
        LDA /FILNAM
        STA SOURCE+$1
```

```
DOITA
       LDY FILNUM          ; POINT TO FILENAME IN DIRECTORY
       JSR PRTFIL          ; GET LOCATION OF T/S
                           ; AND CHECK FILE NAME
       LDA FNDFLG          ; NOT FOUND IF ONE
       BEQ EXIT            ; END OF FILE FOUND
       CLC
       LDA FILNUM          ; POINT TO NEXT FILENAME
       ADC #$23
       STA FILNUM
       BNE DOITA           ; DO AGAIN
                           ;
                           ; WHEN BOTH BYTES OF LINK ARE
                           ; ZERO YOU ARE THROUGH.
                           ;
       LDY #$1             ; GET AND SAVE TRACK
                           ; OF NEXT DIRECTORY SECTOR
       LDA BUFFER,Y
       STA TRACK
       ORA BUFFER+$1,Y     ; IF BOTH TRACK AND SECTOR
                           ; IS ZERO THEN END
       BEQ EXIT
       LDA BUFFER+$1,Y     ; GET SECTOR NUMBER AND SAVE
       STA SECTOR
       JMP DOIT            ; GO DO IT AGAIN
EXIT
       RTS                 ; RETURN
;
PRTFIL:
       LDA BUFFER,Y
       CMP #$FF            ; IS IT A DELETED FILE ?
       BEQ PRTX            ; YES BRANCH
       STA TRACK           ; SAVE TRACK NUMBER OF T/S SECTOR
       INY                 ; SAVE SECTOR NUMBER OF T/S
SECTOR
       LDA BUFFER,Y
       STA SECTOR
       INY
       LDA BUFFER,Y        ; CHECK TYPE OF FILE
       AND #$7F            ; $80 AND $0 IS TEXT FILE
       BNE PRTX            ; NOT TEXT FILE THEN END
       INY                 ; CHECK NAME TO SEE IF CORRECT
       JMP CHKNAM
PRTX:
       RTS                 ; RETURN
;
; MSG PRINTS AN ASCII STRING TO
; THE VIDEO SCREEN.
MSG
```

```
        PLA                     ; PULL RETURN ADDRESS
                                ; TO ZERO PAGE POINTER
        STA ZPAG
        PLA
        STA ZPAG+$1
        JSR INCZ                ; INCREMENT POINTER BY ONE
        STY YSAVE               ; PERSERVE Y REG
        LDY 00
LOOP2
        LDA (ZPAG),Y            ; GET CHAR
        BEQ LOOP3               ; ZERO END STRING
        JSR PUTC                ; OUTPUT TO SCREEN
        JSR INCZ                ; MOVE READ POINTER
        JMP LOOP2               ; DO AGAIN
LOOP3
        JSR INCZ                ; MOVE READ POINTER
        LDY YSAVE               ; RESTORE REG
        JMP (ZPAG)              ; BACK TO PROGRAM
;
YSAVE   HEX 00
ZPAG    EPZ $00
PUTC    EQU $FDED
INCZ
        INC ZPAG                ; INCREMENTS POINTER BY ONE
        BNE INCZ1
        INC ZPAG+$1
INCZ1
        RTS
;
; LINK DISPLACEMENT
;
SOURCE EPZ $06
DEST    EPZ SOURCE+$2
STRCOM
        LDY #$FF                ; COMPARES TWO STRINGS
                                ; USUALLY FILE NAMES
STRCM1
        INY                     ; USED AS POINTER
        DEX                     ; HOLDS LENGTH OF STRING
        BEQ STRCM2              ; DONE THEN BRANCH
        SEC
        LDA (SOURCE),Y          ; CHECK IF EQUAL BY SUBTRACTING
        SBC (DEST),Y
        BEQ STRCM1              ; GET NEXT CHAR
        RTS                     ; RETURN IF NOT EQUAL
;
STRMAB
        LDA #$0                 ; EQUAL RETURN
        RTS
```

```
; HANDLE 256TH COMPARE.
;
STRCM2
        SEC                     ; CHECK LAST CHAR
        LDA (SOURCE),Y
        SBC (DEST),Y
        BNE STRMCA              ; NOT EQUAL THEN BRANCH
STRMAA
        INY                     ; BESURE NOT A SUBSTRING
        CPY #$1E                ; CHECK FOR REST OF SPACES
                                ; $1E MAXIMUM LENGTH OF FILE NAME
        BEQ STRMAB              ; END THEN RETURN
        LDA (DEST),Y            ; CHECK FOR SPACES
        CMP #$A0
        BEQ STRMAA              ; DO AGAIN
        LDA #1                  ; NOT SAME THEN END
STRMCA
        RTS
CHKNAM:                         ; CHECK FILE NAME
        CLC                     ; SET UP FOR ADD
        TYA                     ; UPDATE LOCATION POINTER
        ADC #BUFFER
        STA DEST
        LDA /BUFFER             ; ZERO LOCATION WILL POINT
                                ; TO FILE NAME
        ADC #0
        STA DEST+$1
        LDX LENGTH              ; GET LENGTH
        JSR STRCOM              ; CHECK FILE NAME
        STA FNDFLG              ; ZERO IS FOUND FILE NAME
        JSR SCTNUM              ; GET NUMBER OF SECTORS IN FILE
        RTS
FILNUM HEX 00
FNDFLG HEX 00
ERRFLG HEX 00
SCTNUM:
        CLC
        LDA FILNUM              ; GET POINTER TO FIRST LOCATION
                                ; OF FILE NAME
        ADC #$21                ; ADVANCE TO NUMBER
                                ; OF SECTORS LOCATION
        TAY
        LDA BUFFER,Y            ; GET NUMBER OF SECTORS AND SAVE
        STA SCTCNT
        RTS
RWTS:                           ; READ WRITE TRACK SECTOR ROUTINE
        JSR $BD00               ; READ SECTOR
        BCC RWTSND              ; CARRY SET ERROR OCCURED
        LDY #$D                 ; SET ERROR FLAG
```

```
        STY ERRFLG
        LDA IOB,Y           ; GET ERROR NUMBER
        CMP #$10
        BEQ WRTPRT          ; WRITE PROTECT ERROR OCCURED
        CMP #$20
        BEQ VOLERR          ; VOLUME ERROR OCCURED
        CMP #$40
        BEQ DRVERR          ; DRIVE ERROR OCCURED
        CMP #$80
        BEQ READRR          ; READ ERROR OCCURED
        JSR DSKERR          ; DISPLAY DISK ERROR
        JSR MSG             ; DISPLAY UNDEFINE ERROR
        ASC "UNDEFINED ERROR"
        HEX 8D
        HEX 00
RWTSND
        RTS
DSKERR
        JSR MSG             ; DISPLAY ERROR
        HEX 8D
        ASC "DISK ERROR OCCURED"
        HEX 8D
        HEX 00
        RTS
WRTPRT
        JSR DSKERR          ; DISPLAY ERROR
        JSR MSG             ; DISPLAY WRITE ERROR
        HEX 8D
        ASC "WRITE PROTECT ERROR"
        HEX 8D
        HEX 00
        RTS
VOLERR
        JSR DSKERR
        JSR MSG             ; DISPLAY VOL ERROR
        HEX 8D
        ASC "VOLUME ERROR"
        HEX 8D
        HEX 00
        RTS
DRVERR
        JSR DSKERR
        JSR MSG             ; DISPLAY DRIVE ERROR
        HEX 8D
        ASC "DRIVE ERROR"
        HEX 8D
        HEX 00
        RTS
READRR
```

```
        JSR DSKERR
        JSR MSG             ; DISPLAY READ ERROR
        HEX 8D
        ASC "READ ERROR"
        HEX 8D
        HEX 00
        RTS
RNDFLG  HEX 00
BUFFER  DFS $100
BUFFTS  DFS $100
FILNAM  DFS $20
FILNA:
        END
```

APPENDIX C

COMMODORE 64 TRANSFER PROGRAM LISTINGS

```
20 REM COMMODORE TRANSVERSION DRIVER PROGRAM
40 REM THIS PROGRAM WILL ENABLE TRANSFER ROUTINES OR
   EMULATION ROUTINES
60 IF PEEK(37286)=1 THEN 340:REM SKIP ACKNOWLEDGEMENTS
80 POKE 52,144:POKE56,144:POKE51,0:POKE55,0:CLR
100 REM DISPLAY ACKNOWLEDGEMENTS
120 PRINT CHR$(147);:FOR L=0 TO 10:PRINT:NEXT
140 REM DISABLE FURTHER ACKNOWLEDGEMENTS
160 POKE 37286,1:PRINT"     ";CHR$(18);: PRINT
    "TRANSVERSION" :PRINT
180 REM RESET BINARY RELOCATION POINTER
200 POKE 37285,10:PRINT"THE APPLE TO COMMODORE CONVERSION
    SYSTEM"
220 FOR L=1 TO 8:PRINT:NEXT
240 PRINT"          COPYRIGHT 1986":PRINT:PRINT"BY":PRINT
260 PRINT"  LONALD L. FINK AND THOMAS G. CLEAVER"
280 REM RESERVE MEMORY FOR ROUTINES
300 FOR L=1 TO 5:PRINT:NEXT:FOR L=1 TO 5000: NEXT: PRINT
    CHR$(147)
320 PRINT"COMMODORE DRIVE PROGRAM NOW INSTALLED":PRINT:
    PRINT:GOTO 740
340 IF PEEK(39270)<>0 AND PEEK(37285)<>10 THEN 3220:
    REM PRINT RELOCATION MESSAGE
360 PRINT CHR$(147);
380 REM 'A' IS USED TO BRANCH TO VARIOUS LOCATIONS IN
    PROGRAM AFTER EACH PROGRAM
400 REM LOADS-REMEMBER AFTER EACH LOAD PROGRAM RESTARTS
420 ON A GOTO 1100,2060,1740,1800,2000,1860
440 REM CLEAR POINTERS
460 POKE 37287,0 :POKE 49641,0 :POKE 49642,0 :POKE 49643,0
480 REM A=0 BRANCH;INITIAL VALUE
500 REM CHECKS TO SEE IF EMULATION ROUTINES ALREADY
    INSTALLED
520 REM TRANSFER AND EMULATION PROGRAM USE SAME MEMORY
    LOCATIONS
540 IF PEEK(37525)<>1 OR PEEK(789)=234 THEN 600:
    REM EMULATE NOT INSTALLED
560 REM DISABLING APPLE EMULATION PROGRAM
580 KILL
600 REM RESET RAM MEMORY POINTERS
620 POKE 55,0:POKE 56,144:POKE 51,0:POKE52,144:CLR
640 REM RESERVE MEMORY FOR TRANSFER OR EMULATION ROUTINES
660 PRINT CHR$(147);
680 REM DETERMINE WHAT OPERATOR WANTS TO DO
700 REM EMULATION OR TRANSFER OR EXIT
720 REM DISPLAY QUESTIONS
740 PRINT"PLEASE MAKE SELECTION BY NUMBER.":PRINT:PRINT
760 PRINT"PRESS '1' TO TRANSFER FILE FROM"
780 PRINT"               APPLE TO COMMODORE"
```

```
800 PRINT
820 PRINT"PRESS '2' TO INSTALL APPLE EMULATION"
840 PRINT"                              ROUTINE"
860 PRINT
880 PRINT"PRESS '3' TO EXIT THIS PROGRAM."
900 REM GET ANSWER FROM KEY BOARD
920 REM IGNORE ALL ANSWERS BUT 1 THRU 3
940 GET B$:B=VAL(B$):IF B=0 OR B>3 THEN 940
960 REM JUMP TO PROPER ROUTINE
980 ON B GOTO 1020,1180,2000
1000 REM LOAD TRANSFER ROUTINE SECTION
1020 PRINTCHR$(147):PRINT"LOADING TRANSFERA PROGRAM":
     POKE 37285,0
1040 REM SET A TO JUMP TO LINE 1100
1060 REM LOAD TRANSFER ROUTINE
1080 A=1:LOAD"TRANSFERA",8,1
1100 PRINT"LOADING TRANSFERB PROGRAM"
1120 REM SET A TO JUMP TO LINE 2060
1140 A=2:LOAD"TRANSFERB",8,1
1160 REM LOAD EMULATION ROUTINE SECTION
1180 PRINTCHR$(147);
1200 REM RESET RELOCATION POINTER
1220 POKE 37285,10
1240 REM IS APPLE CHARACTERSET NEEDED "
1280 PRINT
1300 PRINT"PRESS '1' FOR EMULATION ROUTINE ONLY"
1320 PRINT
1340 PRINT"PRESS '2' FOR APPLE CHARACTERSET"
1360 PRINT"             AND EMULATION ROUTINE"
1380 PRINT
1400 PRINT"PRESS '3' TO EXIT PROGRAM."
1420 PRINT
1440 REM IGNORE ALL ANSWERS BUT 1-3
1460 GET B$:B=VAL(B$):IF B=0 OR B>3 THEN 1460
1480 ON B GOTO 1640,1540,2000
1500 REM EMULATION AND APPLE CHARACTER LOAD SECTION
1520 REM LOAD PROGRAMS AND INSTALL ROUTINES
1540 PRINTCHR$(147);
1560 PRINT"LOADING BOOT ALLA PROGRAM"
1580 REM RESET PROGRAM POINTERS AND THEN LOAD PROGRAMS
1600 POKE55,0:POKE51,0:POKE56,144:POKE52,144:CLR:A=0
     :LOAD"BOOTALLA",8,1
1620 REM LOAD EMULATION ROUTINES ONLY SECTION
1640 PRINTCHR$(147);
1660 PRINT"LOADING EMULATIONB PROGRAM"
1680 REM LOAD PROGRAM AND SET POINTER TO LINE 1740
1700 A=3:LOAD"EMULATEB",8,1
1720 REM LOAD PROGRAM AND SET POINTER TO LINE 1800
1740 PRINT"LOADING EMULATIONA PROGRAM"
```

```
1760 A=4:LOAD"EMULATEA",8,1
1780 REM INSTALL EMULATION ROUTINES
1800 A=5:SYS 49152
1820 FORD=0 TO 2000:NEXT D
1840 REM TO UPPER CASE/NORMAL CLEAR SCREEN
1860 PRINTCHR$(146):PRINT CHR$(142):PRINT CHR$(147)
1880 PRINT"EMULATION PROGRAMS LOADED"
1900 PRINT
1920 PRINT"DISPLAYING DIRECTORY":PRINT
1940 REM DISPLAY DIRECTORY
1960 CATALOG
1980 REM END PROGRAM
2000 PRINT:PRINT"PROGRAM EXITED"
2020 END
2040 REM RESERVE MEMORY FOR TRANSFER ROUTINES
2060 POKE 55,0:POKE 56,144:POKE51,0:POKE52,144:CLR
2080 REM GET TYPE OF FILE TO TRANSFER
2100 PRINTCHR$(147) "
2140 PRINT
2160 PRINT"PRESS '1' FOR A BASIC FILE."
2180 PRINT
2200 PRINT"PRESS '2' FOR OTHER TYPE FILES"
2220 PRINT
2240 PRINT"PRESS '3' TO EXIT PROGRAM."
2260 GET B$:B=VAL(B$):IF B=0 OR B>3 THEN 2260
2280 ON B GOTO 2340,3040,2000
2300 REM TRANSFER BASIC PROGRAM SECTION
2320 REM GET TYPE OF TRANSFER OPTIONS
2340 PRINTCHR$(147)
2360 PRINT "YOU WISH TO TRANSFER A BASIC PROGRAM !!"
2380 PRINT
2420 PRINT"DO YOU WISH APPLE CHARACTERSET OPTION" "
2460 PRINT
2480 PRINT"PRESS '1' FOR APPLE CHARACTERSET LINE."
2500 PRINT
2520 PRINT"PRESS '2' FOR NO APPLE CHARACTERSET LINE"
2540 PRINT
2560 PRINT"PRESS '3' TO EXIT PROGRAM."
2580 PRINT
2600 GET B$:B=VAL(B$):IF B=0 OR B>3 THEN 2600
2620 ON B GOTO 2660,2700,2000
2640 REM SET APPLE CHARACTERSET FLAG
2660 POKE 49642,255
2665 REM IF CHARACTERSET AUTOMATICALLY
2666 REM INVOKES EMULATION LINES
2670 GOTO 3020
2680 REM CHECK FOR EMULATION OPTION
2700 PRINT CHR$(147)
2720 PRINT "YOU WISH TO TRANSFER A BASIC PROGRAM !!"
```

```
2740 PRINT
2760 PRINT"DO YOU WISH EMULATION OPTION " "
2800 PRINT
2820 PRINT"PRESS '1' FOR EMULATION PROGRAM"
2840 PRINT"                    OPTION LINES"
2860 PRINT
2880 PRINT"PRESS '2' FOR NO EMULATION OPTION LINES"
2900 PRINT
2920 PRINT"PRESS '3' TO EXIT PROGRAM."
2940 PRINT
2960 GET B$:B=VAL(B$):IF B=0 OR B>3 THEN 2960
2980 ON B GOTO 3020,3040,2000
3000 REM SET EMULATION FLAGS
3020 POKE 49643,255:POKE 49641,255
3040 PRINTCHR$(147):REM CLEAR SCREEN
3060 REM JUMP TO TRANSFER ROUTINE
3080 REM DO TRANSFER
3100 SYS 38144
3101 END
3120 REM CLEAR VARIABLES
3140 CLR
3160 REM CHECK FOR RELOCATION OF BINARY FILE
3180 IF PEEK(39270)=0 THEN 360:REM BACK TO MENU
3200 REM DISPLAY RELOCATION WARNING
3220 PRINT
3240 PRINT"WARNING BINARY PROGRAM WAS RELOCATED"
3260 PRINT:PRINT"TO BASIC PROGRAM AREA"
3280 PRINT "STARTING AT MEMORY LOCATION 2049"
3300 PRINT
3320 PRINT"PRESS ANY NUMBER FOR MENU"
3340 B$="9"
3360 GET B$:B=VAL(B$):IF B=0 THEN 3360
3380 REM RESET RELOCATION FLAG
3400 POKE 39270,0
3420 REM BACK TO MENU
3440 GOTO 340
3460 END
```

```
5  REM BOOT ALLA PROGRAM
10 PRINTCHR$(147)
20 PRINT"LOADING CHARACTERSET PROGRAM"
30 IFA=0THEN A=1:LOAD"CHARACTERSET",8,1
40 PRINT CHR$(147)
50 PRINT"INSTALLING CHARACTERSET"
60 IF A=1 THENA=2:SYS 36880
70 PRINTCHR$(147)
80 PRINT"LOADING 'EMULATEB' PROGRAM"
90 IF A=2 THEN A=3:LOAD"EMULATEB",8,1
100 PRINT CHR$(147)
110 PRINT"LOADING 'EMULATEA' PROGRAM"
120 IF A=3 THEN A=4:LOAD"EMULATEA",8,1
130 PRINTCHR$(147)
140 PRINT"INSTALLING APPLE EMULATE"
150 SYS 49152
160 TEXT
170 HOME
180 PRINTCHR$(146);:PRINTCHR$(142);:PRINTCHR$(147);
190 FOR A=32 TO 127:PRINT CHR$(A);:NEXTA
200 PRINT:PRINT
210 PRINT"NEW APPLE CHARACTERSET PRINTED"
220 FOR B=0 TO 900:NEXTB
230 A=6
240 LOAD"MENU",8
250 END
```

```
; THIS PROGRAM IS NAMED SEQRNDA
; BEGINNING OF TRANSFER PROGRAM
; CHAINS TO ASSY.TRANSFERC AND
; ASSY.TRANSFERD
; WILL TRANSFER SEQUENTIAL AND RANDOM ACCESS
; BINARY AND BASIC TYPE FILES
; ROM ROUTINES USED
        ACPTR=$FFA5
       CHROUT=$FFD2        ; DISPLAY CHAR IN A REGISTER
       SCREEN=1300         ; A SCREEN LOCATION
           FA=$BA
           SA=$B9
         TALK=$FFB4        ; SEND TALK TO IEEE BUS
         TKSA=$FF96        ; SEND SECONDARY ADDRESS
        UNTLK=$FFAB        ; SEND UNTALK TO IEEE
       FCLOSE=$FFC3        ; CLOSE FILE IN A REGISTER
        FOPEN=$FFC0        ; OPEN FILE
       STATUS=$90          ; STATUS LOCATION
       CHKOUT=$FFC9        ; SAME AS BASIC'S CMD
        CLRCH=$FFCC        ; RESET DEFAULT DEVICES
          TMP=$FB
       SETLFS=$FFBA        ; SET FILE PARAMETERS
       SETNAM=$FFBD        ; SET NAME PARAMETERS
       SAVPGM=$FFD8        ; SAVE PRGRAM ROUTINE
       LODPGM=$FFD5        ; LOAD A FILE ROUTINE
        BSOUT=$FFD2        ; DISPLAY A CHAR
        BASIN=$FFCF        ; GET A CHARACTER
       LINPRT=$BDCD        ; PRINT TWO BYTE HEX NUMBER
         DDRB=$DD03        ; DATA DRECTION
        PORTB=$DD01        ; RS232 I/O ADDRESS
       PGRMST=$2B          ; PGRAM STRT PTR
       *=$9500             ; ORIGIN
STARTA
       JSR MSG
       .BYTE 147,13,'TRANSFER PROGRAM'
       .BYTE ' WAITING ON APPLE.'
       .BYTE 13,13,'CONTINUE WITH APPLE'
       .BYTE ' PROGRAM!!!!!',13,13,00
       LDA #20
       STA AMOUT           ; SET DELAY RATE
       LDY #0
       STY FLAG
       LDA #6
       STA DDRB            ; SETUP PORTB
       LDX #0
       STX PORTB
       LDY #$FF
NAME
       INY                 ; GET FILE NAME
```

```
        JSR CHAR
        STA FILNAM,Y      ; SAVE NAME
        STA 1401,Y
        BNE NAME
        STY LENGTH        ; SAVE LENGTH
        JSR CHAR          ; GET TYPE FILE
        STA TYPE
        CMP #'M           ; BINARY FILE TYPE ?
        BEQ BIN
        CMP #'S           ; SEQUENTIAL TEXT FILE ?
        BEQ SEQTA
        CMP #'R           ; RANDOM ACCESS ?
        BEQ RANDO
        CMP #'B
        BEQ BASICA
        JSR MSG
        .BYTE 147,'TRANSFER PROCEDURE ERROR',13
        .BYTE 'OR TRANSFER I/O ERROR',13
        .BYTE 13,'ABORTING PROGRAM',13,0
        RTS
BIN
        JMP BINARY
RANDO
        JMP RANDOM
SEQTA
        JMP SEQTAL
BASICA
        JSR MSG           ; BASIC TRANSFER PROGRM
        .BYTE 147,'TRANSFERING BASIC FILE',13,13,00
        JMP BASIC
SEQTAL
        JSR MSG
        .BYTE 147,'TRANSFERING SEQUENTIAL TEXT
FILE',13,13,00
        JMP SEQTLA
RANDOM
        JSR MSG
        .BYTE 147,'TRANSFERING RANDOM ACCESS FILE',13,13,00
        JMP RANDMA
BINARY
        JSR MSG
        .BYTE 147,'TRANSFERING BINARY FILE',13,13,00
        JSR CHAR          ; GET START ADDRESS
        STA LOCSTR        ; SAVE LOW BYTE
        STA POINT
        STA PGRMST
        JSR CHAR
BINA
        STA LOCSTR+1      ; GET HI BYTE
```

```
        STA PGRMST+1
        STA POINT+1         ; CHECK TO SEE IF TO RELOCATE
; RELOCATE WHEN START IS BELOW
; $400 HEX OR GREATER THAN
; $9500 HEX
        CLC
        LDA POINT+1
        CMP #4
        BEQ BINB            ; OKAY DONT RELOCATE
        BCS BINB            ; OKAY DONT RELOCATE
BIND
        LDA #1
        STA NOFLAG          ; SET RELOCATE FLAG
        STA LOCSTR          ; SAVE LOW BYTE
        STA POINT
        STA PGRMST
        LDA #8              ; RELOCATE
        BNE BINA
BINB
        LDA POINT+1
        CMP #$94
        BEQ BINC            ; OKAY
        BCC BINC            ; OKAY
        LDA #8              ; NO - RELOCATE
        BNE BIND
LLTEMP
        .BYTE 00,00
BINC
        JSR CHAR
        STA NDRESS          ; SAVE LENGTH BYTE
        STA LLTEMP
        JSR CHAR            ; GET LENGTH HI BYTE
        STA NDRESS+1        ; AND SAVE
        STA LLTEMP+1
BINH
        CLC
        LDA NDRESS          ; GET END LO ADRESS
        ADC PGRMST
        STA NDRESS
        LDA NDRESS+1        ; GET END HI BYTE
        ADC PGRMST+1
        STA NDRESS+1        ; CHECK TO SEE IF TO RELOCATE
; RELOCATE WHEN END ADDRESS IS
; GREATER THAN $9500 HEX
; ABORT IF NOFLAG EQUALS ONE
; THAT IS ALREADY RELOCATED
; BUT PROGRAM TO LONG > $9100 HEX
        CLC
        LDA NDRESS+1
```

```
        CMP #$95
        BEQ BINE
        BCS BINE
        BCC BINF
BINE
        LDA NOFLAG
        BNE BING
        LDA #1
        STA PGRMST
        STA NOFLAG
        STA LOCSTR          ; SAVE LOW BYTE
        STA POINT
        LDA #8
        STA PGRMST+1
        STA LOCSTR+1        ; GET HI BYTE
        STA POINT+1
        LDA LLTEMP
        STA NDRESS
        LDA LLTEMP+1
        STA NDRESS+1
        LDA #1
        BNE BINH
BING
        JSR MSG
        .BYTE 147,13,'BINARY PROGRAM TO LONG'
        .BYTE ' !!!!'
        .BYTE 13,13,'PRESS 'RETURN' FOR MENU.'
        .BYTE 13,13,0
ANS
        JSR BASIN
        CMP #13
        BNE ANS
        JMP FSAVEA
BINF
        LDX #0
DISNAM
        LDA NAMDIS,X
        BEQ DISTRT
        JSR BSOUT
        INX
        BNE DISNAM
NAMDIS
        .BYTE 13,13
        .BYTE 13,'BINARY FILE NAME IS: '
FILNAM
        .BYTE 'FILENAME'
        *=*+32
        .BYTE 00
DISTRT
```

```
        TAX                     ; DISPLAY START MESS
        JSR MSG
        .BYTE 13,13
        .BYTE 'BINARY START ADDRESS'
        .BYTE ' IS : ',00
        LDX LOCSTR              ; GET START LOCATION
        LDA LOCSTR+1
        JSR LINPRT              ; DISPLAY TO SCREEN
        LDX #0                  ; DISPLAY END MESSAGE
        JSR MSG
        .BYTE 13,13
        .BYTE 'BINARY END ADDRESS IS : '
        .BYTE 00
                                ; DISPLAY END LOCATION
        LDX NDRESS
        LDA NDRESS+1
        JSR LINPRT
        JSR MSG
        .BYTE 147,'BINARY FILE TRANSFER HAS BEGUN.',13,0
        LDX #0
GETA
        LDY #0                  ; USED AS COUNTER
LOOPD
        JSR CHAR                ; GET NEXT DIGIT
        STA (PGRMST),Y          ; STORE VALUE
        JSR CHECK
        LDA FLAG
        BNE DONEB
        JSR MOVEB
        BNE LOOPD               ; ALWAYS GET NEXT VALUE
MOVEB                           ; MOVE PROGRAM POINTER TO
        INC POINT
        INC PGRMST              ; NEXT BASIC CHARACTER
        BNE MVERTN
        INC POINT+1
        INC PGRMST+1
MVERTN
        RTS
CHAR
        SEI                     ; DISABLE INTERRUPTS
        CLC                     ; CLEAR CARRY
        STY YTMP                ; PERSERVE REGS
        STX XTMP
        LDX #8                  ; SET NUMBER OF BITS
        LDA #6
        STA PORTB               ; ENABLE TRANSMSION
        JSR DELAYA              ; SMALL DELAY 1/2 BIT
STRBIT
        LDA PORTB               ; GET DATA
```

```
        LSR A               ; LOOK FOR
        BCS STRBIT          ; STARTBIT
        JSR DELAYA          ; GET TO MIDDLE OF BIT
        NOP                 ; MORE TIMING
        NOP
BITLUP
        JSR DELAY           ; DELAY ROUTINE
        LDA PORTB           ; GET NEXT BIT
        LSR A               ; MOVE BIT TO CARRY
        ROR VALUE           ; SHIFT TO STORAGE
        BIT $0D             ; TIMING
        NOP
        DEX
        BNE BITLUP          ; ALL DONE ?
        STX PORTB           ; YES DSABL TRNSMSN
        JSR DELAY           ; TWO STOP BITS DELAY
        JSR DELAY
        LDX XTMP            ; RETREIVE REGS
        LDY YTMP
        LDA VALUE           ; GET VALUE OF RECEIVED
        CLI                 ; DATA
        RTS                 ; RETURN TO MAIN PROGRAM
; DELAYS COUNTER TIMES 7+15CYCLES
DELAY
        JSR DELAYA          ; 91 CYCLES +85 CYCLES
DELAYA
        STY YTMPA           ; BAUD RATE DELAY
        LDY #0              ; ROUTINE PERSERVE Y
DLY
        INY                 ; TOTAL OF 85 CYCLES
        CPY AMOUT
        BNE DLY
        LDY YTMPA           ; RETREIVE Y REG
        RTS
DONEB                       ; UPDATE END OF PROGRAM
        JSR MSG
        .BYTE 147,'BINARY TRANSFER COMPLETE.',13,13
        .BYTE 'SAVING BINARY FILE TO DISK.',13,13,00
        LDA NOFLAG          ; ORIGINAL ADDRESS ?
        BEQ FSAVE           ; YES SAVE
        LDA PGRMST          ; GET NEW END ADDRESS
        STA NDRESS
        LDA PGRMST+1
        STA NDRESS+1
        LDA #1              ; SET NEW START ADDRESS
        STA PGRMST          ; RESTORE BASIC POINTER
        STA LOCSTR          ; SET NEW LOW BYTE
        LDA #$08
        STA PGRMST+1        ; RESTORE BASIC POINT
```

```
        STA LOCSTR+1        ; SET HIGH BYTE
FSAVE                       ; SAVE TO DISK
        CLC
        LDA NDRESS          ; INCREASE END ADDRESS
        ADC #1              ; BY ONE
        STA NDRESS
        LDA NDRESS+1
        ADC #0
        STA NDRESS+1
        LDA #1              ; LOGICAL FILE NUMBER
        LDX #8              ; DEVICE NUMBER OF FLOPPY
        LDY #0              ; SECONDARY ADDRESS
        JSR SETLFS
        LDA LENGTH          ; GET LENGTH OF NAME
        LDX #<FILNAM        ; ADDRESS OF NAME
        LDY #>FILNAM        ; HIGH BYTE
        JSR SETNAM
        LDA LOCSTR          ; GET START ADDRESS
        STA TMP
        LDA LOCSTR+1        ; GET HIGH BYTE
        STA TMP+1
        LDA #<TMP           ; POINT TO ADDRESS
        LDX NDRESS
        LDY NDRESS+1
        JSR SAVPGM
        BCC FSAVEA
        JMP $E0F9
FSAVEA
        LDA #$00
        STA $800
        TAY
        TAX
        LDA #1
        STA PGRMST          ; RESTORE POINTER
        LDA #8
        STA PGRMST+1        ; TO BASIC START
        LDA #8              ; DO A NEW EFFECTIVELY
        STA $2E
        LDA #3
        STA $2D
        TYA
        STA ($2B),Y
        INY
        STA ($2B),Y
        JSR $A68E
        TXA
        JSR $FFE7
        LDA $37
        LDY $38
```

```
        STA $33
        STY $34
        LDA $2D
        LDY $2E
        STA $2F
        STY $30
        STA $31
        STY $32
        JSR $A81D
        LDX #$19
        STX $16
        JMP LODBAS           ; LOAD AND RUN MENU
        RTS
CHECK                        ; CHECK FOR END OF PROGRAM
        LDA POINT+1          ; HIGH BYTE
        CMP NDRESS+1
        BNE CHKRTN           ; NO-NOT DONE
        LDA POINT            ; LOW BYTE ?
        CMP NDRESS
        BNE CHKRTN           ; NO-NOT DONE
        LDA #1               ; YES SET DONE FLAG
        STA FLAG
CHKRTN
        RTS
POINT
        .WORD 00
NOFLAG
        .BYTE 00
AMOUT
        .BYTE 00
TYPE
        .BYTE 00
FLAG
        .BYTE 00
RCLGTH
        .BYTE 00
VALUE
        .BYTE 00
YTMPA
        .BYTE 00
LOCSTR
        .WORD 00
NDRESS
        .WORD 0000
LENGTH
        .BYTE 00
OPNSEQ
        LDA LENGTH
        LDX #<FILNA
```

```
        LDY #>FILNA
        JSR SETNAM
        LDY #0
        STY STATUS
        LDA #$02
        LDX #$08
        LDY #$02
        JSR SETLFS
        JSR FOPEN
        LDX STATUS
        BNE OPNERR
        LDX #$02
        JSR CHKOUT
OPNERR
        RTS
FILNA
        .BYTE '@0:'
        *=*+$2A
SEQTLA
        LDY #$FF
SEQTLB
        INY
        LDA FILNAM,Y
        STA FILNA+3,Y
        CPY LENGTH
        BNE SEQTLB
        LDA #',
        STA FILNA+3,Y
        INY
        LDA #'S
        STA FILNA+3,Y
        INY
        LDA #',
        STA FILNA+3,Y
        INY
        LDA #'W
        STA FILNA+3,Y
        CLC
        LDA LENGTH
        ADC #7
        STA LENGTH
        JSR OPNSEQ
        LDA STATUS
        BNE SEQEND
SEQSND
        JSR CHAR
        CMP #0
        BEQ SEQNDA
        JSR BSOUT
```

```
        LDA STATUS
        BEQ SEQSND
SEQEND
        JSR GETERR
SEQNDA
        JSR CLRCH
        LDA #$2
        JSR FCLOSE
        JSR MSG
        .BYTE 147
        .BYTE 'SEQUENTIAL FILE TRANSFER COMPLETED',13,13,00
        RTS
RANDMA
        JSR CHAR             ; GET RECORD LENGTH
        STA RCLGTH           ; SAVE
        LDY #$FF
RANDMB
        INY
        LDA FILNAM,Y         ; GET FILENAME
        STA FILNB+2,Y
        CPY LENGTH
        BNE RANDMB
        LDA #',              ; ADD RECORD LENGTH
        STA FILNB+2,Y
        INY
        LDA #'L
        STA FILNB+2,Y
        INY
        LDA #',
        STA FILNB+2,Y
        INY
        LDA RCLGTH           ; RECORD LENGTH
        STA FILNB+2,Y
        INC RCLGTH           ; ADD 1 TO RECORD LENGTH
        CLC
        LDA LENGTH           ; ADD 6 FILENAME LENGTH
                             ; FOR DISK COMMANDS
        ADC #6
        STA LENGTH
        LDA #0               ; INITIALIZE RECORD COUNTER TO 1
        STA RECORD+1
        STA STATUS
        LDA #1
        STA PSTION
        STA RECORD           ; OPEN COMMAND CHANNEL TO DISK
        LDA #15              ; SET FILE PARAMETERS
        LDX #8
        LDY #15
        JSR SETLFS
```

```
        LDA #0              ; SET NAME PARAMETERS
        JSR SETNAM
        JSR FOPEN           ; OPEN FILE
        LDX STATUS          ; CHECK IF OKAY
        CPX #50             ; RECORD NOT PRESENT
        BEQ RANDMC          ; ERROR IS OKAY
        CPX #0
        BNE RRRRRR          ; BRANCH ERROR
                            ; OPEN RELATIVE FILE
RANDMC
        LDY #0              ; CLEAR STATUS
        STY STATUS
        LDA #$02            ; SET FILE PARAMETERS
        LDX #$08
        LDY #$02
        JSR SETLFS
        LDA LENGTH          ; SET NAME PARAMETERS
        LDX #<FILNB
        LDY #>FILNB
        JSR SETNAM
        JSR FOPEN           ; OPEN RELATIVE FILE
        LDX STATUS          ; CHECK STATUS
        CPX #50             ; RECORD NOT PRESENT
        BEQ RANDME          ; ERROR IS OKAY
        CPX #0
RRRRRR
        BNE END             ; ERROR BRANCH
RANDME
        LDX STATUS          ; CHECK STATUS
        CPX #50             ; RECORD NOT PRESENT
        BEQ LOOPE           ; ERROR IS OKAY
        CPX #0              ; ZERO NO ERROR OCCURRED
        BNE END             ; ERROR BRANCH
                            ; WRITE POINTER TO ZERO
LOOPE
        LDY #0              ; ZERO END COUNTER
        STY FLAG
        JSR COMMND          ; GET RECORD AND POSITION
                            ; FROM APPLE SET UP TO
                            ; WRITE TO DISK
LOOPG
        JSR CHAR            ; GET CHARACTER FROM APPLE
        CMP #0
        BNE LOOPF           ; IF NOT ZERO VALID DATA
        INC FLAG            ; ZERO WAS SENT
        LDA FLAG            ; IS THIS THE END OF TRANSFER
        CMP #4
        BNE LOOPE           ; NO JUST END OF RECORD BRANCH
        JMP END             ; YES END
```

```
LOOPF
       JSR BSOUT           ; WRITE CHAR TO DISK
       LDX STATUS          ; CHECK STATUS
       CPX #50
       BEQ RANDMD
       CPX #0
       BNE END             ; ERROC OCCURED THEN ABORT
RANDMD
       LDY PSTION          ; UPDATE POSITION COUNTER
       INY
       CPY RCLGTH          ; CHECK IF END OF RECORD
       BEQ MOVRCD          ; YES THEN CHANGE RECORD NUMBER
       STY PSTION          ; NO JUST POSITION UPDATE
       JMP LOOPG           ; GET NEXT CHAR
MOVRCD
       INC RECORD          ; INCREASE RECORD COUNTER BY ONE
       BNE MOVRCE
       INC RECORD+1
MOVRCE
       LDA #1              ; SET POSITION TO FIRST BYTE OF
                           ; RECORD
       STA PSTION
       JSR COMMNE          ; SEND "POSITION COMMAND" TO DISK
       JMP LOOPG           ; GET NEXT CHARACTER
END
       LDA STATUS          ; CHECK STATUS
       CMP #0
       BEQ ENDA            ; NO ERROR THEN BRANCH
       JSR GETERR          ; DISPLAY ERROR
ENDA
       JSR CLRCH           ; RESET DEFAULT I/O DEVICES
       LDA #2              ; CLOSE RELATIVE FILE
       JSR FCLOSE
       LDA #15             ; CLOSE COMMAND CHANNEL
       JSR FCLOSE
       JSR MSG             ; DISPLAY MESSAGE
       .BYTE 147,13,13,'RANDOM ACCESS'
       .BYTE ' FILE TRANSFER COMPLETED.'
       .BYTE 13,13,00
       RTS
FILNB
       .BYTE '0:'
       *=*+$30
LOCATE
       .BYTE 00            ; GETS RECORD AND POSITION
                           ; FROM APPLE
                           ; SEND POSITION COMMAND TO DISK
                           ; SET OUTPUT DEVICE AS
                           ; THE RELATIVE FILE
```

```
                                ; EXPAND FILE IF NEEDED
COMMND
       STA ATMP                 ; SAVE REGISTERS
       STY YTMP
       STX XTMP
       JSR CHAR                 ; GET RECORD FROM APPLE --LOW
BYTE
       STA RECORD               ; SAVE TO RECORD COUNTER--LOW
BYTE
       BNE COMMNB
       INC FLAG                 ; IF ZERO INCREMENT END FLAG
COMMNB
       JSR CHAR                 ; GET RECORD HIGH BYTE
       STA RECORD+1             ; SAVE
       BNE COMMNC
       INC FLAG                 ; IF ZERO INCREMENT END FLAG
COMMNC
       JSR CHAR                 ; GET POSITION FROM APPLE
       STA PSTION               ; AND SAVE
       BNE COMMNE
       INC FLAG                 ; IF ZERO CHECK END FLAG
       LDA FLAG                 ; GET FLAG
       CMP #3                   ; THREE ZERO OCCURED
       BEQ COMEND               ; YES THEN ABORT
COMMNE
       LDA #0                   ; CHECK STATUS
       STA STATUS
COMMNG
       LDX #15                  ; SET RELATIVE FILE FOR WRITE
       JSR CHKOUT
       LDX STATUS               ; CHECK STATUS
       CPX #50
       BEQ COMMNH               ; RECORD NOT PRESENT ERROR OKAY
       CPX #0
       BNE COMEND               ; ERROR BRANCH-ABORT
COMMNH
       LDY #0                   ; SEND POSITION COMMAND
                                ; TO DISK DRIVE
       STY COMLOC
CAMNDA
       LDY COMLOC               ; GET COMMAND LOCATION
       LDA CMMND,Y              ; GET COMMAND CHARACTER
       JSR BSOUT                ; SEND CHARACTER TO DISK DRIVE
       LDX STATUS               ; CHECK STATUS
       CPX #50                  ; RECORD NOT PRESENT ERROR OKAY
       BEQ COMMNI
       CPX #0
       BNE COMEND               ; ERROR BRANCH-ABORT
COMMNI
```

```
        LDY COMLOC          ; GET COMMAND POINTER
        INY                 ; UPDATE AND SAVE
        CPY #6
        STY COMLOC          ; CHECK FOR END
        BEQ COMNDB          ; YES DONE BRANCH
        BNE CAMNDA          ; NOT DONE CONTINUE
COMNDB
        JSR CLRCH           ; RESET DEFAULT I/O DEVICES
        LDX #2              ; SET RELATIVE FILE AS CURRENT
                            ; OUTPUT DEVICE
        JSR CHKOUT
        LDX STATUS          ; GET STATUS
        CPX #50             ; RECORD NOT PRESENT ERROR
        BNE COMEND          ; NO THEN END
        LDA #255            ; YES WRITE TO RECORD
        JSR BSOUT           ; TO REMOVE ERROR
        JSR CLRCH           ; RESET DEFAULT DEVICES
        JMP COMMNE          ; RESET POSITION
COMEND
        LDA ATMP            ; RESTORE REGISTERS
        LDY YTMP
        LDX XTMP
        RTS
CMMND
        .BYTE 80
        .BYTE 98
RECORD
        .WORD 0000          ; RECORD COUNTER
PSTION
        .BYTE 00            ; POSITION POINTER
        .BYTE 13
ATMP
        .BYTE 00            ; TEMP STORAGE
YTMP
        .BYTE 00
XTMP
        .BYTE 00
COMLOC
        .BYTE 00
GETERR                      ; GET DRIVE ERROR
        LDA STATUS
        CMP #64             ; IGNORE IF END OF FILE
        BNE GTRYB
        RTS
GTRYB
        JSR CLRCH           ; RESET DEFAULT DEVICES
        LDA #13             ; OUTPUT LF TO SCREEN
        JSR CHROUT
        LDA #8              ; DEV # OF DISK
```

```
        STA FA
        JSR TALK          ; SEND TALK
        LDA #$6F          ; SEND SEC ADDRESS
        STA SA
        JSR TKSA          ; SEND SEC FOR TALK
GTRYA
        JSR ACPTR         ; READ ERROR BYTE
        JSR CHROUT        ; DISPLAY TO SCREEN
        CMP #13
        BNE GTRYA         ; DO UNTIL DONE
        RTS               ; DONE
SAVBAS                    ; SAVE BASIC PROGRAM
        LDA #1            ; SET FOR SAVE
        LDX #8            ; TO DISKETTE
        LDY #0            ; SEC ADDRESS =0
        JSR SETLFS        ; SET LOGICAL FILE
        LDA LENGTH        ; GET FILENAM LENGTH
        LDX #<FILNAM      ; POINT TO FILENAME
        LDY #>FILNAM
        JSR SETNAM        ; SET FILE NAME PARAM.
        LDX $2D           ; END ADDRESS OF BASIC
        LDY $2E
        LDA #$2B          ; START ADDRESS OF BASIC
        JSR SAVPGM
        BCC SVEBAS        ; CLEAR IS NO ERROR
        JMP $E0F9         ; PRINT ERROR
SVEBAS
        RTS
LODBAS                    ; LOAD MENU PROGRAM
                          ; MUST JUMP TO THIS ROUTINE
                          ; NOT JUMP SUBROUTINE
        JSR MSG
        .BYTE 147,'LOADING MENU PROGRAM.',13,0
        LDA #1            ; LOGICAL FILE NUMBER
        STA PGRMST
        LDX #8            ; DEVICE NUMBER OF DISK
        STX PGRMST+1
        LDY #1            ; DEFAULT SEC ADDRESS
        JSR SETLFS        ; SET FILE PARAMETERS
        LDA #4            ; LENGTH OF NAME
        LDX #<MENU        ; POINT TO NAME
        LDY #>MENU
        JSR SETNAM        ; SET FILE NAME
        LDA #0
        LDX $2B
        LDY $2C
        JSR LODPGM
        BCC LDBASA
        JMP $E0F9
```

```
LDBASA
        JSR $FFB7           ; GET STATUS
        AND #$BF            ; CLEAR EOF BIT
        BEQ LDBASB          ; NO ERROR BRANCH
        JSR GETERR
        JMP $E19C
LDBASB
        STX $2D
        STY $2E
LDBASC
        LDA #1
        STA PGRMST
        LDX #8
        STX PGRMST+1
        LDA #0
        JSR $A533
        PLA
        STA STKSVE
        PLA
        STA STKSVE+1
        PLA
        STA STKSVA
        PLA
        STA STKSVA+1
        LDA #0
        JSR $A871
        LDA STKSVA+1
        PHA
        LDA STKSVA
        PHA
        LDA STKSVE+1
        PHA
        LDA STKSVE
        PHA
        RTS
STKSVE
        .WORD 0000
STKSVA
        .WORD 0000
MENU
        .BYTE 'MENU',0
        .FIL ASSY.TRANSFERC
        .END
```

```
; THE PROGRAM FILE NAME IS ' ASSY.TRANSFERC'
          ZPAG=$FB
        OLDEND=$14
        NEWEND=$24
           TMP=$FB
        LNSTRT=$FD
         BSOUT=$FFD2
          DDRB=$DD03        ; DATA DRECTION
         PORTB=$DD01        ; RS232IOADDRSS
        PGRMST=$2B          ; PGRAM STRT PTR
        PGRMND=$2D          ; PGRM END PTR
          TEMP=$22
        STRING=$26
         BASIN=$FFCF
LINE
        LDA #8              ; TO START OF BASIC PRGRAM
        STA TEMP+1
        LDA #0
        STA TEMP
LINEA
        JSR MOVE            ; SKIP PAST LINE LINKS
        JSR MOVE
        JSR MOVE            ; GET  LS BYTE OF LINE #
LINEAA
        STA LSBLNR
        LDX TEMP            ; SAVE ADDRESS OF LINE #
        STX LNSTRT
        LDX TEMP+1
        STX LNSTRT+1
        JSR MOVE            ; GET MS BYTE OF LINE #
        JSR LINEZ           ; GET LENGTH OF LINE #
        JSR MOVE
        LDX TEMP            ; SAVE ADDRESS OF FIRST
        STX TMP             ; TOKEN OF LINE
        LDX TEMP+1
        STX TMP+1
LINEB
        JSR SUMADD          ; GET NEW SUM OF CHAR
        LDA SUM             ; GREATER THAN 80 ?
        CMP #81
        BCS SPLIT           ; YES SPLIT LINE
        JSR MOVE            ; NO GET NEXT CHAR
        BEQ LINEC           ; END OF LINE BRANCH
        BNE LINEB           ; DO IT AGAIN
LINEC                       ; END OF PROGRAM ?
        INY
        LDA (TEMP),Y
        BEQ LINED           ; MAYBE CHECK AGAIN
        BNE LINEA           ; NO GET NEW LINE
```

```
LINED
        INY
        LDA (TEMP),Y        ; END OF PROGRAM ?
        BNE LINEA           ; NO GET NEW LINE
        RTS                 ; YES RETURN
SPLIT
        LDA TEMP            ; SAVE ADDRESS AT WHICH
        STA ENDST           ; SUM IS GREATER THAN
        LDA TEMP+1          ; 80
        STA ENDST+1
        JSR NEW             ; GET NEXT LINE NUMBER
        LDY #0
SPLITA
        LDX TMP             ; REACH LOCATION WHERE
        CPX ENDST           ; SUM IS GREATER THAN 80
        BNE SPLITM          ; NO CONTINUE
        LDX TMP+1           ; STILL CHECKING ?
        CPX ENDST+1
        BNE SPLITM          ; NO CONTINUE
SPLITC                      ; MOVE TO END OF LINE
        JSR MOVE
        BNE SPLITC
        JMP LINEA           ; GO TO NEXT LINE
SPLITM                      ; CHECK ON COLON
        LDA (TMP),Y         ; GET CHARACTER
        CMP #':             ; IS IS A COLON ?
        BNE SPLITO          ; NO CHECK IF TOKEN
        BEQ SPLTMB
SPLTMA
        LDA #0              ; END LINE WITH ZERO
        TAY
        STA (TMP),Y
        LDA #4              ; ADD 4 LOCATION AND NEW
        STA AMOUNT          ; LINE NUMBER
        JSR AMOVE
        LDA TMP             ; TELL WHERE TO ADD FOUR
        STA TEMP            ; SPACES
        LDA TMP+1
        STA TEMP+1
        JSR ADD             ; ADD 4 SPACES
        JSR MOVE            ; MOVE TO NEXT LOCATION
        JSR MOVE            ; SKIP LINE LINKS
        LDA NUMSTR          ; GET LINE NUMBER
        STA (TEMP),Y        ; STORE LS BYTE
        INY
        LDA NUMSTR+1        ; STORE MS BYTE
        STA (TEMP),Y
        DEY
        LDA (TEMP),Y
```

```
        JMP LINEAA          ; GET NEXT LINE
SPLTMB
        CLC
        LDY #0
        LDA (LNSTRT),Y      ; GET CURRENT LINE
        ADC #1              ; NUMBER ADD 1
        STA NUMSTR          ; AND SAVE
        INY
        LDA (LNSTRT),Y
        ADC #0
        STA NUMSTR+1
        DEY
        LDA NUMSTR+1        ; CHECK LINE NUMBER
        CMP NEWEND+1        ; SMALLER THAN ?
        BCC SPLTMA          ; YES SPLIT
        BEQ SPLTMC          ; KEEP CHECKING
        BCS SPLITC          ; YES SPLIT
NUMSTR
        .WORD 0000
SPLTMC                      ; CURRENT LINE NUMBER SMALLER
        LDA NUMSTR          ; THAN NEXT LINE NUMBER ?
        CMP NEWEND
        BCC SPLTMA          ; YES SPLIT
        BCS SPLITC          ; NO DON'T SPLIT
SPLITO
        CMP #139            ; IS IT A 'IF' TOKEN ?
        BEQ SPLITC          ; YES  NO SPLIT
        CMP #143            ; REM TOKEN ?
        BEQ SPLITC          ; YES NO SPLIT
        JSR AMOVE           ; MOVE NEXT LOCATION
        JMP SPLITA          ; DO AGAIN
LSBLNR
        .BYTE 0
MSBLNR
        .BYTE 0
LINEZ
        STA MSBLNR          ; FIND NUMBER OF
        CMP #0              ; DIGITS IN LINE NUMBER
        BEQ LOW             ; MSB ZERO IE <255
        SEC                 ; NO FIND VALUE >255
        LDA LSBLNR          ; GET LSBYTE OF LINE #
        SBC GRAND           ; COMPARE WITH 1000
        STA SUM             ; SAVE RESULT
        LDA MSBLNR          ; GET MSBYTE OF LINE #
        SBC GRAND+1         ; COMPARE WITH 1000
        ORA SUM             ; SET FLAGS
        BEQ FOUR            ; SET SUM EQUAL 4
        BCC THREE           ; SET EQUAL 3
        LDA LSBLNR
```

```
        SBC TENGRD          ; COMPARE WITH 10000
        STA SUM
        LDA MSBLNR
        SBC TENGRD+1        ; COMPARE WITH 10000
        ORA SUM             ; SET FLAGS
        BEQ FIVE            ; SET SUM EQUAL FIVE
        BCC FOUR            ; SET SUM EQUAL FOUR
        BCS FIVE            ; SET SUM EQUAL FIVE
LOW
        LDA LSBLNR
        CMP #100
        BCS THREE
        CMP #10
        BCS TWO
        LDA #1
RETURN
        STA SUM
        RTS
NEW                         ; FIND NEXT LINE NUMBER
        JSR MOVE            ; GO TO END OF LINE
        BNE NEW
NEWA
        LDY #3              ; GET NEXT LINE NUMBER
        LDA (TEMP),Y
        STA NEWEND          ; AND SAVE
        INY
        LDA (TEMP),Y
        STA NEWEND+1
        LDA ENDST           ; RESTORE POINTER
        STA TEMP
        LDA ENDST+1
        STA TEMP+1
        LDY #0
        RTS
SUM
        .BYTE 00
FIVE
        LDA #5
        BNE RETURN
FOUR
        LDA #4
        BNE RETURN
THREE
        LDA #3
        BNE RETURN
TWO
        LDA #2
        BNE RETURN
GRAND
```

```
        .BYTE 232
        .BYTE 3
TENGRD
        .BYTE 16
        .BYTE 39
SUMADD
        CMP #128
        BCC SUMDDB
        SBC #128               ; CALCULATE CODE
        TAX                    ; USED AS POINTER
        LDA SUMTBL,X
SUMDDA
        CLC
        ADC SUM
        STA SUM
        RTS
SUMDDB
        LDA #1
        BNE SUMDDA
SUMTBL
        .BYTE 3,3,4,4,6,5,3,4,3,4
        .BYTE 3,2,7,5,6,3,4,2,4,4
        .BYTE 4,6,3,4,6,5,4,4,3,3,4
        .BYTE 4,5,3,3,4,2,2,4,4,3,4
        .BYTE 1,1,1,1,1,3,2,1,1,1,3,3,3
        .BYTE 3,3,3,3,3,3,3,3,3,3,3,4,3
        .BYTE 4,3,3,4,5,6,4,0,4,4,5,4,4
        .BYTE 5,7,3,5,3,3,4,5,3,5,5,4,4
        .BYTE 1,4,5,4,3


        *=$C000                ; ORIGIN
BASIC
        LDA #20
        STA AMOUT              ; SET DELAY RATE
        LDA #$01
        STA PGRMST             ; SET PROGRAM START
        LDY #0
        STY TEMP               ; CLEAR FLAGS
        STY FLAG
        LDA #$08
        STA PGRMST+1
        STA TEMP+1
        LDA #6
        STA DDRB               ; SETUP PORTB
        LDX #0
        STX PORTB              ; DISABLE TRANS
        STX CKLINE             ; INITIALIZE
```

```
       LDA #204          ; DEFAULTS TO NO
       STA TBLCDE        ; EMULATION
       LDA EMUFLG        ; NO EMUL - ZERO
       BEQ LN2END        ; YES- NON ZERO
TEXTA3
       LDA TEXT3,Y
       CMP #$FF
       BEQ TEXTB3
       STA (PGRMST),Y
       INY
       BNE TEXTA3
TEXTB3 LDA CFLAG         ; WHAT CHAR SET ?
       BEQ LN0END        ; NO SKIP
TEXTA0
       LDA TEXT0,X       ; YES $FF STRNG
       CMP #$FF          ; END OF STRING ?
       BEQ LN0END
       STA (PGRMST),Y    ; WRITE CHAR SET LN
       INX
       INY               ; 0 IN BASIC PROGRAM MEMORY
       BNE TEXTA0
LN0END
       LDA GFLAG         ; WANT GRAPHICS SET
       BEQ LN1END        ; NO SKIP IT
       LDX #0
TEXTA1
       LDA TEXT1,X       ; GET CHAR
       CMP #$FF          ; END OF LINE ?
       BEQ LN1END        ; YES SKIP
       STA (PGRMST),Y    ; NO WRITE CHAR
       INX               ; OR TOKEN
       INY               ; MOVE TO NEXT CHAR
       BNE TEXTA1        ; ALWAYS BRANCH
LN1END
       LDA #215          ; SET FLAG FOR EMULATION
       STA TBLCDE
       LDX #0            ; POINT TO FIRST CHAR
TEXTA2
       LDA TEXT2,X       ; GET CHAR
       CMP #$FF          ; END OF LINE ?
       BEQ LN2END        ; YES BRANCH
       STA (PGRMST),Y    ; STORE CHARACTER
       INX               ; GO TO NEXT CHARACTER
       INY
       BNE TEXTA2        ; ALWAYS BRANCH
LN2END
       TYA               ; GET NUMBER OF CHARACTERS
       CLC               ; STORED
       ADC PGRMST        ; GET NEW START
```

```
        STA PGRMST          ; LOCATION
        LDA PGRMST+1
        ADC #0
        STA PGRMST+1
        LDX #0
        JSR MSG
        .BYTE 147,'BASIC PROGRAM'
        .BYTE ' TRANSFER HAS BEGUN'
        .BYTE $0D,00


        LDY #0              ; USED AS COUNTER
LOOP
        JSR CHAR            ; GET FIRST DIGIT
        JSR CONVRT
        STA (PGRMST),Y      ; STORE VALUE
        BNE LOOPA
        INC FLAG            ; CHECK FOR END OF PGRM
        LDA FLAG
        CMP #3
        BNE LOOPB
        BEQ DONE            ; END OF TRANSFER
LOOPA
        LDA #0
        STA FLAG
LOOPB
        JSR MOVEB
        BNE LOOP            ; ALWAYS GET NEXT VALUE
DONE                        ; UPDATE END OF PROGRAM
        CLC                 ; POINTER
        LDA PGRMST
        ADC #1
        STA PGRMND
        LDA PGRMST+1
        ADC #0
        STA PGRMND+1
        LDA #1              ; RESTORE PROGRAM START
        STA PGRMST          ; POINTER
        LDA #8              ; TO START OF BASIC
        STA PGRMST+1
        JSR $A81D           ; RESTORE LINE LINKS
        JSR $A533
        JSR MSG
        .BYTE 147,'BASIC TRANSFER COMPLETED',$0D,$0A
        .BYTE 'PROCESSING DISK COMMANDS'
        .BYTE $0D,$0A,$00
        JMP PRINT           ; PROCESS DISK COMMANDS
```

```
; SUBROUTINE TO CONVERT APPLE TOKEN
; TO COMMODORE TOKEN
; ALL NON CONVERTABLE TOKENS TO
; TO ASCII STRINGS
CONVRT
       STX XTMP            ; PERSERVE REGS
       STY YTMP
       INC CKLINE          ; PLACE IN LINE
       LDX CKLINE          ; 1-4 LOCATIONS
       CPX #5              ; NEED NO CONVERSION
       BNE SPCRTA
       PHA                 ; SAVE DATA IN A
       LDA #$20            ; ADD SPACE TO START OF
       STA (PGRMST),Y      ; LINE
       JSR MOVEB           ; MOVE TO NEXT CHAR
       PLA                 ; RETREIVE TOKEN OR CHAR
SPCRTA
       CPX #5
       BCS TOKEN           ; LINE NUMBER
       BCC GOOD            ; TO END ROUTINE
TOKEN
       CMP #0              ; CHECK FOR E O LINE
       BNE TABLE           ; END LINE IS ZERO
       STA CKLINE          ; ZERO LINE POINTER
       LDA PGRMST          ; GET IN OF LINE LOC
       STA TEMP            ; SAVE LSB
       LDA PGRMST+1        ; GET EOL LOC MSB
       STA TEMP+1          ; SAVE
       LDA #0
       BEQ GOOD
TABLE
       TAX                      ; SAVE CODE
       SEC
       SBC #132            ; SMALLER THAN 132
       BCS TBLA            ; BRANCH IF LARGER
       TXA                 ; GOOD NO CONVERSION GET CODE
       BNE GOOD            ; ALWAYS BRANCH
TBLA
       TAX                 ; CODE IN A 0-102
       LDA CODE,X          ; GET NEW CODE
       CMP TBLCDE          ; CHECK FOR INVALID ?
       BCS BDCODE          ; CODE YES BRANCH
GOOD
       LDX XTMP            ; GOOD CODE
       LDY YTMP            ; RESTORE REGISTERS
       CMP #0              ; SET FLAGS
       RTS
BDRTN
       LDY EMUFLG
```

```
        CPY #0              ; EMULATE MODE IS NON ZERO
        BEQ BDRTB           ; NO EMULATE BRANCH
        CLC                 ; YES EMULATE
        ADC #12             ; POINT TO EMULATE STRING
        TAX
        BNE INSERT          ; SKIP REMOUT
BDCODE
        SBC #204            ; GET INDEX 0-51
        ASL A               ; 0 TO 204
        TAX
        CMP #22             ; CHECK FOR NOTRACE TO
        BCC BDRTB           ; LOWER DO AS USUAL
        CMP #32             ; LOMEM CODES
        BCS BDRTNA          ; HIGHER DO AS USUAL
        BCC BDRTN           ; YES ADJUST POINTER
BDRTNA
        LDY EMUFLG
        CPY #0              ; EMULATE MODE ?
        BEQ BDRTA           ; NO BRANCH
        CMP #32             ; YES CHECK HCOLOR CODE
        BEQ BDRTNB
        CMP #70             ; YES CHECK HGR2 CODE
        BEQ BDRTNC
        CMP #44             ; CHECK PDL CODE
        BEQ BDRTND          ; YES CHANGE POINTER
        BNE BDRTA
BDRTNB
        LDX #46             ; POINT HCOL STRING
        BNE INSERT
BDRTNC
        LDX #48             ; POINT HGR2 STRING
        BNE INSERT
BDRTND
        LDX #100
        BNE INSERT
BDRTA
        CMP #94             ; IF "NOT" CODE
        BEQ INSERT          ; SKIP REMOUT
BDRTB
        LDY #5
        LDA #143            ; REM LINE AT BEGINNING
        STA (TEMP),Y
INSERT
        LDY #0
        STY COUNTR          ; RESET LENGTH COUNTER
        LDA LCTBLE,X        ; FIND STRING ADDRESS
        STA STRING          ; SAVE STRING ADDRESS
        LDA LCTBLE+1,X      ; GET HI BYTE
        STA STRING+1
```

```
        TXA                     ; GET POINTER CODE 0-102
        LSR A                   ; GET LENGTH POINTER 0-51
        TAX                     ; USED AS POINTER
        LDA LENTBL,X
        STA LNGTH               ; GET LENGTH AND SAVE
OKAY
        LDY COUNTR              ; START AT BEGINING
        LDA (STRING),Y          ; GET NEXT CHAR
        INY
        STY COUNTR              ; UPDATE COUNTER
        CPY LNGTH               ; END OF STRING
        BEQ RESET               ; YES DONE
        LDY YTMP                ; NO STORE CHAR
        STA (PGRMST),Y
        JSR MOVEB               ; MOVE TO NEXT CHAR
        BNE OKAY                ; ALWAYS
RESET
        JMP GOOD
LNGTH
        .BYTE 00
COUNTR
        .BYTE 00                ; POINTER FOR STRNG
CKLINE
        .BYTE 00                ; LOC IN LINE PNTER
EMUFLG
        .BYTE 00                ; EMULATION FLAG
CFLAG
        .BYTE 00                ; FLAG FOR CHAR SET
GFLAG
        .BYTE 00
TBLCDE
        .BYTE 00                ; FLAGS NVALID CODE
TEXT3
        .BYTE 24,8,0,0,66,178,194
        .BYTE '(788)',58,151,32,51,55,50,56,55,44,48,0,$FF
TEXT0
        .BYTE 57,8,1,0,139,65,178,48,167,65,178,49,58,147
        .BYTE '"CHARACTERSET",8,1',0
        .BYTE 8,8,2,0,139,65,178,49,167,65,178,50,58,158,
              32,51,54,56,56,48
        .BYTE 0,$FF
TEXT1
        .BYTE 113,8,3,0,139,66,179,177,52,53,175,65,179,51,
              167,65,178,51,58
        .BYTE 147,'"EMULATEB",8,1',0,$FF
TEXT2
        .BYTE 148,8,4,0,139,66,179,177,52,53,175,65,178,51,
              167,65,178,52,58
        .BYTE 147,'"EMULATEA",8,1',0,159,8,5,0,158
```

```
        .BYTE '49152',0,$FF
LENTBL
        .BYTE 6,6,7,6,6
        .BYTE 7,9,5,7,5
        .BYTE 5,9,8,8,8
        .BYTE 8,9,2,2,3
        .BYTE 2,2,5,3,2
        .BYTE 7,8,5,4,5
        .BYTE 5,6,6,6,6
        .BYTE 6,6,7,6,8
        .BYTE 8,8,8,7,3
        .BYTE 6,4,2,7,6
        .BYTE 1,14
LCTBLE
        .WORD HTAB
        .WORD HOME
        .WORD TRACE
        .WORD VTAB
        .WORD TEXT
        .WORD FLASH
        .WORD NVERSE
        .WORD HGR
        .WORD HPLOT
        .WORD POP
        .WORD GIT
        .WORD TROFF
        .WORD SPEED
        .WORD NRMAL
        .WORD LOMEM
        .WORD HIMEM
        .WORD HCOLR
        .WORD ATROFF
        .WORD ASPEED
        .WORD ANRMAL
        .WORD ALOMEM
        .WORD AHIMEM
        .WORD PDL
        .WORD AHCOLR
        .WORD AHGR2
        .WORD ERR
        .WORD RESUME
        .WORD DEL
        .WORD GR
        .WORD PR
        .WORD IN
        .WORD CALL
        .WORD PLOT
        .WORD HLIN
        .WORD VLIN
```

```
         .WORD HGR2
         .WORD DRAW
         .WORD XDRAW
         .WORD ROT
         .WORD SCALE
         .WORD SHLOAD
         .WORD COLOR
         .WORD RECALL
         .WORD STORE
         .WORD SGN
         .WORD POKE
         .WORD AT
         .WORD NOT
         .WORD SCRN
         .WORD PEEK
         .WORD APDL
         .WORD NVALID
TEMPA
         .WORD 0000
HTAB
         .BYTE '_HTAB_'
HOME
         .BYTE '_HOME_'
TRACE
         .BYTE '_TRACE_'
TROFF
         .BYTE '_NOTRACE_'
VTAB
         .BYTE '_VTAB_'
TEXT
         .BYTE '_TEXT_'
SPEED
         .BYTE '_SPEED=_'
FLASH
         .BYTE '_FLASH_'
NRMAL
         .BYTE '_NORMAL_'
NVERSE
         .BYTE '_INVERSE_'
LOMEM
         .BYTE '_LOMEM:_'
HIMEM
         .BYTE '_HIMEM:_'
HCOLR
         .BYTE '_HCOLOR=_'
HGR
         .BYTE '_HGR_'
HPLOT
         .BYTE '_HPLOT_'
```

```
POP
        .BYTE '_POP_'
GIT
        .BYTE '_GET_'
PDL
        .BYTE '_PDL_'
ERR
        .BYTE '_ONERR_'
RESUME
        .BYTE '_RESUME_'
DEL
        .BYTE '_DEL_'
GR
        .BYTE '_GR_'
PR
        .BYTE '_PR#_'
IN
        .BYTE '_IN#_'
CALL
        .BYTE '_CALL_'
PLOT
        .BYTE '_PLOT_'
HLIN
        .BYTE '_HLIN_'
VLIN
        .BYTE '_VLIN_'
HGR2
        .BYTE '_HGR2_'
DRAW
        .BYTE '_DRAW_'
XDRAW
        .BYTE '_XDRAW_'
ROT
        .BYTE '_ROT=_'
SCALE
        .BYTE '_SCALE=_'
SHLOAD
        .BYTE '_SHLOAD_'
COLOR
        .BYTE '_COLOR=_'
RECALL
        .BYTE '_RECALL_'
STORE
        .BYTE '_STORE_'
SGN
        .BYTE '_&_'
POKE
        .BYTE '_POKE_'
AT
```

```
        .BYTE '_AT_'
NOT
        .BYTE '0='
SCRN
        .BYTE '_SCRN(_'
PEEK
        .BYTE '_PEEK_'
ATROFF
        .BYTE 168,215
ASPEED
        .BYTE 216,178
ANRMAL
        .BYTE 217,176,217
ALOMEM
        .BYTE 218,58
AHIMEM
        .BYTE 219,58
AHCOLR
        .BYTE 220,176,178
AHGR2
        .BYTE 211,50
APDL
        .BYTE 226
NVALID
        .BYTE '_INVALID CODE_'
CODE
        .BYTE 133,231,134,135
        .BYTE 232,208,233,234,235
        .BYTE 236,237,238,239,211
        .BYTE 220,212,240,241,204
        .BYTE 205,242,243,244,206
        .BYTE 215,217,210,209,245
        .BYTE 213,207,219,218,229
        .BYTE 230,246,247,216,136
        .BYTE 137,138,139,140,38
        .BYTE 141,142,143,144,145
        .BYTE 146,147,148,150,249
        .BYTE 153,154,155,156,214
        .BYTE 162,163,164,165,166
        .BYTE 167,250,251,169,170
        .BYTE 171,172,173,174,175
        .BYTE 176,177,178,179,180
        .BYTE 181,182,183,184,252
        .BYTE 226,185,186,187,188
        .BYTE 189,190,191,192,193
        .BYTE 253,195,196,197,198
        .BYTE 199,200,201,202,255
        .BYTE 255,255,255,255,255
        .BYTE 255,255,255,255,255
```

```
        .BYTE 255,255,255,255,255
        .BYTE 255,255,255,255,255
SRHVAL
        .BYTE 00
FOUND
        .BYTE 00
DECFLG
        .BYTE 00
AMOUNT
        .BYTE 00

; MSG PRINTS AN ASCII STRING TO
; THE VIDEO SCREEN.
MSG
        PLA
        STA ZPAG
        PLA
        STA ZPAG+$1
        JSR INCZ
        STY YSAVE
        LDY #00
LOOP2
        LDA (ZPAG),Y
        BEQ LOOP3
        JSR BSOUT
        JSR INCZ
        JMP LOOP2
LOOP3
        JSR INCZ
        LDY YSAVE
        JMP (ZPAG)
;
YSAVE
        .BYTE 00
INCZ
        INC ZPAG
        BNE INCZ1
        INC ZPAG+$1
INCZ1
        RTS
        .FIL ASSY.TRANSFERD
        .END
```

```
;PROGRAM FILE NAME IS 'ASSY.TRANSFERD'
START
        LDA PGRMST          ; GET START POINT
        STA TEMP            ; SAVE USED AS POINTER
        LDA PGRMST+1
        STA TEMP+1
        DEC TEMP
SEARCH
        CLC
        LDA TEMP            ; SKIP LINE NUMBERS
        ADC #5              ; AND ADDRESS POINTERS
        STA TEMP
        STA LNSTRT
        LDA TEMP+1
        ADC #0
        STA TEMP+1
        STA LNSTRT+1
        RTS
BEGIN
        LDA #0
        TAY
        STA FOUND           ; CLEAR FOUND FLAG
        LDA TEMP            ; CHECK FOR END OFPROGRAM
        CMP PGRMND
        BNE BEGINA
        LDA TEMP+1
        CMP PGRMND+1
        BCS ENDB
BEGINA
        LDA (TEMP),Y        ; GET CHARACTER
        BEQ EOLINE          ; END OF LINE ?
        CMP SRHVAL          ; NO IS IT THIS ONE ?
        BEQ DONEA
        INC TEMP            ; NO TRY AGAIN
        BNE BEGIN
        INC TEMP+1
        BNE BEGIN           ; ALWAYS BRANCH
EOLINE
        INY
        LDA (TEMP),Y        ; END OF PROGRAM ?
        BNE SEARH           ; NO TRY AGAIN
        INY
        LDA (TEMP),Y        ; END OF PROGRAM ?
        BNE SEARH           ; NO TRY AGAIN
        BEQ ENDB            ; YES DONE
SEARH
        JSR SEARCH
        LDY #0
        BEQ BEGINA          ; ALWAYS BRANCH
```

```
DONEA
        LDA #1              ; SET FOUND FLAG
        STA FOUND           ; TMP POINTS TO ADDRESS
        LDA TEMP
        STA TMP
        LDA TEMP+1
        STA TMP+1
ENDB
        LDY #0
        RTS
CHRDNC
        LDA EMUFLG
        BNE CHRA
        JMP CHRDNF
CHRA
        JSR START           ; TO BASIC START
        LDA #161            ; CHANGE GET TO GIT
        STA SRHVAL
        JSR BEGIN
        LDA FOUND
        BEQ CHRDNA
        LDA #214
        STA (TMP),Y
        BNE CHRDNC
CHRDNA
        LDA #223            ; CATALOG TOKEN
        STA SRHVAL
        LDA #188            ; INSERT LOG TOKEN
        STA STRVAL
        JSR CHRDNB
        JMP CHRDNF
CHRDNB
        JSR START
CHRBBB
        JSR BEGIN
        LDA FOUND
        BEQ CHRDND
        JSR MOVE
        LDA #1
        STA AMOUNT
        JSR ADD
        LDA STRVAL
        STA (TEMP),Y
        BNE CHRBBB
CHRDND
        RTS
STRVAL
        .BYTE 00
CHRDNG
```

```
        STA SRHVAL
        JSR START
        JSR SPACEA
        RTS
CHRDNF
        JSR SPACE
        LDA #'5
        JSR CHRDNG
        LDA #'8
        JSR CHRDNG
        LDA #212
        JSR CHRDNG
        LDA #14
        JSR CHRDNG
        LDA #15
        JSR CHRDNG
        LDA #'4
        JSR CHRDNG
        LDA #16
        JSR CHRDNG
        LDA #17
        JSR CHRDNG
        JSR TOKCHG
        LDA #',
        JSR CHRDNG
        JSR LINE            ; SPLIT LINES
        JSR MSG
        .BYTE 147,'SAVING BASIC PROGRAM'
        .BYTE 13,00
        JSR $A533           ; SET LINE LINKS
        JSR SAVBAS
        JMP LODBAS
CHRDNE
        JMP CHRDNC
PRINT
        LDA #0              ; CLEAR FLAGS
                            ; NOMON DEFAULT
        STA MNCFLG          ; SET POINTER TO
        JSR START           ; START OF PROGRAM
        LDA #153            ; PRINT TOKEN IS
        STA SRHVAL          ; CHARACTER TO FIND
PRINTA
        JSR BEGIN           ; SEARCH FOR CHARACTER
        LDA FOUND           ; FOUND IT ?
        BEQ CHRDNE          ; NO BRANCH TO END
PRINTB
        JSR MOVE            ; TO NEXT VALUE
        CMP #$20            ; IS IT A SPACE ?
        BEQ PRINTB          ; YES  SKIP IT
```

```
        CMP #'D             ; IS IT D IN D$
        BEQ PRINTD
        CMP #199            ; IS IS A CHR$ TOKEN
        BEQ PRINTE
        CMP #'"             ; IS IT "
        BEQ PRINTF
        BNE PRINTA          ; NO TRY AGAIN
PRINTF
        JSR MOVE            ; MOVE TO FIRST CHAR
        CMP #$20            ; INSIDE QUOTE SKIPPING
        BEQ PRINTF          ; SPACES
        CMP #4              ; REM CONTROL D
        BNE PRINTA          ; NOT DISK COMMAND
        BEQ DISK            ; YES PROCESS
PRINTE
        JSR MOVE            ; SKIP SPACES
        CMP #$20
        BEQ PRINTE
        CMP #'(
        BNE PRINTA          ; NOT DISK TRY AGAIN
PRNTEE
        JSR MOVE
        CMP #$20            ; SKIP SPACES
        BEQ PRNTEE
        CMP #'4             ; LOOK FOR (4)
        BNE PRINTA          ; NO TRY AGAIN
PRNTEF
        JSR MOVE
        CMP #$20            ; SKIP SPACES
        BEQ PRNTEF
        CMP #')             ; LOOK FOR (4)
        BNE PRINTA          ; NO TRY AGAIN
        BEQ COMAND
COMNDA
        JMP APPEND
PRINTD
        JSR MOVE
        CMP #'$
        BNE PRINTA
COMAND                      ; CHECKS FOR QUOTE
        JSR MOVE            ; REMOUT NO QUOTES
        BEQ COMNDA
        CMP #'"             ; IS IT A QUOTE
        BNE COMAND          ; NO SKIP IT
DISK
                            ; CHECKS FOR DISK
        JSR MOVE            ; COMMANDS
        LDA TEMP            ; SAVE LOCATION OF
        STA TEMPA           ; BEGINNING OF DISK
```

```
        LDA TEMP+1          ; COMMAND
        STA TEMPA+1
        LDX #0
DSK
        LDA CMDTBL,X        ; CHECK WITH TABLE
        BMI LASTCH          ; GET LAST LETTER
        CMP (TEMP),Y        ; ARE THEY THE SAME ?
        BNE NXTWRD          ; NO- GOTO NEXT WORD
        JSR MOVE            ; YES GET NEXT LETTER
        INX                 ; MOVE POINTER
        BNE DSK             ; CHECK NEXT LETTER
LASTCH
                            ; STRIP MSB OFF VALUE
        AND #$7F            ; OFF OF LAST LETTER
        CMP (TEMP),Y        ; IS IT THE SAME ?
        BNE NXTWRE          ; NO TRY NEXT WORD
        INX                 ; YES GET ADDRESS
        LDA CMDTBL+1,X      ; M S BYTE
        PHA                 ; USED AS RETURN ADDRESS
        LDA CMDTBL,X        ; GET LS BYTE
        PHA                 ; SAVE ON STACK
        RTS
                            ; EXECUTE COMMAND
NXTWRD                      ; GOES TO NEXT COMMAND LOC
        INX                 ; POINT TO NEXT LETTER
        LDA CMDTBL,X        ; GET NEXT LETTER
        BPL NXTWRD          ; END OF WORD ?
NXTWRE                      ; YES SKIP ADDRESS
        INX                 ; INCREMENT POINTER TO NEXT
        INX                 ; WORD LOCATION
        INX                 ; POINTING TO FIRST LETTER
        LDA CMDTBL,X
        BEQ ENDTBL          ; END OF TABLE BRANCH
        LDA TEMPA           ; RESTORE BEGINNING OF
        STA TEMP            ; DISK COMMAND WORD
        LDA TEMPA+1
        STA TEMP+1
        BNE DSK             ; TRY NEXT WORD ALWAYS
ENDTBL
        JMP PRINTA          ; TRY AGAIN
MOVE
        LDY #0              ; MOVE POINTER
        INC TEMP
        BNE MOVEA
        INC TEMP+1
MOVEA
        LDA (TEMP),Y        ; GET PRESENT CHAR
        RTS
DECMNT                      ; DECREMENTS POINTER TO
```

```
        LDA #0              ; ADD CHARACTER TO PROGRAM
        STA DECFLG          ; RESET FINISH FLAG
        DEC NEWEND          ; DECREASE NEW END
        LDA NEWEND          ; OF BASIC POINTER
        CMP #$FF
        BNE DECMTA
        DEC NEWEND+1
DECMTA
        DEC OLDEND          ; DECREASE PREVIOUS
        LDA OLDEND          ; BASIC END POINTER
        CMP #$FF
        BNE DECMTB
        DEC OLDEND+1
DECMTB
        LDA TEMP            ; CHECK TO SEE IF
        CMP OLDEND          ; WE REACHED THE
        BNE DECMTC          ; LOCATION WHERE WE ARE
        LDA TEMP+1          ; INSERTING CHARACTERS
        CMP OLDEND+1
        BNE DECMTC
        LDA #1              ; SET FLAG IF WE ARE DONE
        STA DECFLG
DECMTC
        RTS
ADD
        CLC
        LDA PGRMND          ; CALCULATE TO END LOC
        STA OLDEND          ; AND SAVE LOCATION
        ADC AMOUNT
        STA PGRMND          ; RESET BASIC POINTERS
        STA NEWEND          ; SAVE NEW END LOCATION
        LDA PGRMND+1
        STA OLDEND+1
        ADC #0
        STA PGRMND+1
        STA NEWEND+1
        LDY #0
ADDA
        LDA (OLDEND),Y      ; GET CHAR IN OLD
        STA (NEWEND),Y      ; LOC AND TRANSFER
        JSR DECMNT          ; TO NEW LOCATION
        LDA DECFLG          ; DECREMENT POINTERS
        BEQ ADDA            ; AND CHECK IF DONE
        LDA (OLDEND),Y      ; GET LAST CHAR
        STA (NEWEND),Y
        LDA #$20            ; ADD SPACES IN LINE
ADDB
        STA (OLDEND),Y
        INY
```

```
        CPY AMOUNT          ; ADD CORRECT AMOUNT ?
        BNE ADDB            ; NO DO AGAIN
        LDY #0
        RTS
SPACE
        JSR START           ; SET POINTERS
        LDA #'"             ; LOOK FOR "
        STA SRHVAL
SPACEA
        JSR BEGIN
        LDA FOUND           ; IS IT FOUND ?
        BEQ SPCDNE          ; NO THEN BRANCH
SPACEB
        INY
        LDA (TMP),Y         ; FIND  # OF SPACES
        CMP #$20
        BEQ SPACEB
        STY YTMP            ; SAVE NUMBER OF SPACES
SPACEC
        LDY #1
        STA (TMP),Y         ; STORE CHAR AFTER
        JSR AMOVE           ; SPACES
        CLC
        LDA TMP             ; GET LOCATION
        ADC YTMP
        CMP PGRMND          ; END OF PROGRAM ?
        BEQ SPACEE
SPACED
        LDY YTMP            ; GET NEXT VALUE
        LDA (TMP),Y
        JMP SPACEC
SPACEE
        CLC
        LDA TMP
        ADC YTMP
        LDA TMP+1
        ADC #0
        CMP PGRMND+1        ; END OF PROGRAM ?
        BNE SPACED          ; NO THEN BRANCH
        LDY YTMP
        LDA (TMP),Y
        LDY #1
        STA (TMP),Y
        DEC YTMP
        SEC
        LDA PGRMND          ; RESET END OF PROGRAM
        SBC YTMP            ; BY NUMBER OF SPACES
        STA PGRMND          ; DELETED
        LDA PGRMND+1
```

```
        SBC #0
        STA PGRMND+1
        JSR MOVE            ; GET NEXT LOCATION
        JMP SPACEA          ; GO FIND MORE SPACES
SPCDNE
        RTS
CMDTBL
        .BYTE 'OPE',$CE     ; OPEN
        .WORD OPEN-1
        .BYTE 'CLOS',$C5    ; CLOSE
        .WORD CLOSE-1
        .BYTE 'REA',$C4     ; READ
        .WORD READ-1
        .BYTE 'WRIT',$C5    ; WRITE
        .WORD WRITE-1
        .BYTE 'APPEN',$C4 ; APPEND
        .WORD APPEND-1
        .BYTE 'POSITIO',$CE ; POSITION
        .WORD APPEND-1
        .BYTE 'EXE',$C3     ; EXEC
        .WORD EXEC-1
        .BYTE 'VERIF',$D9 ; VERIFY
        .WORD APPEND-1
        .BYTE 'MO',$CE
        .WORD MON-1
        .BYTE 'NOMO',$CE
        .WORD NOMON-1
        .BYTE 'DELET',$C5
        .WORD DELETE-1
        .BYTE 'LOA',$C4
        .WORD APPEND-1
        .BYTE 'SAV',$C5
        .WORD SAVE-1
        .BYTE 'RU',$CE
        .WORD RUN-1
        .BYTE 'BRU',$CE
        .WORD BRUN-1
        .BYTE 'BSAV',$C5
        .WORD BSAVE-1
        .BYTE 'BLOA',$C4
        .WORD BLOAD-1
        .BYTE 'MAXFIL',$C5
        .WORD APPEND-1
        .BYTE 'CATALO',$C7
        .WORD CATALG-1
        .BYTE 00            ; END OF TABLE
TMPST
        .WORD 0000
CHARCT
```

```
        .BYTE 00
BRNFLG
        .BYTE 00
BSVFLG
        .BYTE 00
BLDFLG
        .BYTE 00
ENDST
        .WORD 0000
BTOK
        LDA #222            ; GET B TOKEN
        LDY #0
        STA (TMP),Y         ; WRITE TO BASIC LINE
        JSR AMOVE           ; MOVE WRITE POINTER
        RTS
BRUN
        LDY #0
        STY BSVFLG          ; BSAVE FLAG
        STY BLDFLG          ; BLOAD FLAG
        STY CHARCT
        LDX #1
        STX BRNFLG          ; BRUN FLAG
        BNE EXECAA
BSAVE
        LDY #0
        STY BRNFLG          ; BRUN FLAG
        STY BLDFLG          ; BLOAD FLAG
        STY CHARCT
        LDX #1
        STX BSVFLG          ; BSAVE FLAG
        BNE EXECAA
BLOAD
        LDY #0
        STY BRNFLG          ; BRUN FLAG
        STY BSVFLG          ; BSAVE FLAG
        STY CHARCT
        LDX #1
        STX BLDFLG          ; BLOAD FLAG
        BNE EXECAA
EXEC
        LDY #0
        STY CHARCT
        STY BRNFLG          ; BRUN FLAG
        STY BSVFLG          ; BSAVE FLAG
        STY BLDFLG          ; BLOAD FLAG
EXECAA  LDA EMUFLG          ; EMULATE ?
        BNE EXECA           ; YES BRANCH
        JMP APPEND          ; REM LINE
EXECA
```

```
       LDA MNCFLG          ; MON IN EFFECT
       BEQ EXECF           ; NO BRANCH
       LDA TEMP            ; UPDATE WRITE POINTER
       STA TMP             ; FROM READ POINTER
       LDA TEMP+1
       STA TMP+1
EXECAB
       JSR AMOVE           ; MOVE WRITE PNTER
       INC CHARCT          ; GET NUMBER OF CHAR
       LDA (TMP),Y         ; GET CHARACTER
       BEQ EXECB           ; MOVE TO END OF LINE
       CMP #':
       BEQ EXECB
       BNE EXECAB          ; CONTINUE TO END
EXECB
       INC CHARCT          ; ADD 4 MORE
       INC CHARCT
       INC CHARCT
       INC CHARCT
       LDA TEMP            ; SAVE READ POINTER
       PHA
       LDA TEMP+1
       PHA
       LDA TMP             ; SETUP POINTER TO ADD
       STA TEMP            ; SPACES FOR DATA
       LDA TMP+1           ; AT WRITE LOCATION
       STA TEMP+1
       LDA CHARCT          ; GET NUMBER OF SPACES
       STA AMOUNT
       JSR ADD
       PLA                 ; RESTORE READ POINTER
       STA TEMP+1
       PLA
       STA TEMP
       LDY #0
       LDA #':             ; ADD COLON
       STA (TMP),Y
       JSR AMOVE           ; MOVE WRITE POINTER
EXECF
       LDA BRNFLG          ; BRUN COMMAND ?
       BEQ EXECFA          ; NO BRANCH
       JSR BTOK            ; YES WRITE B TOKEN
       LDA #138            ; WRITE RUN TOKEN
       BNE EXECFD          ; ALWAYS
EXECFA LDA BSVFLG          ; BSAVE COMMAND ?
       BEQ EXECFB          ; NO BRANCH
       JSR BTOK
       LDA #148            ; SAVE TOKEN
       BNE EXECFD          ; ALWAYS
```

```
EXECFB
        LDA BLDFLG         ; BLOAD COMMAND
        BEQ EXECFC         ; NO BRANCH
        JSR BTOK
        LDA #147
        BNE EXECFD
EXECFC
        LDA #221           ; ADD EXEC TOKEN
EXECFD
        STA (TMP),Y
        JSR AMOVE
        LDA #$20           ; ADD SPACE
        STA (TMP),Y
EXCFDA
        JSR MOVE           ; MOVE READ POINTER
        CMP #$20           ; SKIP SPACES
        BEQ EXCFDA
        CMP #'"            ; "EXEC" FORM
        BEQ EXECD
EXECC
        JSR AMOVE          ; WRITE FILENANE
        STA (TMP),Y
        JSR MOVE
        CMP #'"            ; END OF FILENAME
        BNE EXECC
EXECD
        JSR MOVE
        BEQ EXECE          ; LOOK FOR END OF LINE
        CMP #':
        BEQ EXECE
        CMP #',
        BNE EXECDA
        JSR MOVE
EXECDA
        CMP #';
        BNE EXECDB
        JSR MOVE
EXECDB
        CMP #0
        BEQ EXECE
        CMP #':
        BEQ EXECE
        JSR AMOVE
        STA (TMP),Y        ; WRITE CHAR
        JSR MOVE
        JMP EXECDA
EXECE
        JMP CLOSEA
                           ; END OF LINE
```

```
                            ; WITH SPACES
OPEN
       LDA MNCFLG           ; MON IN EFFECT ?
       BNE OPERTN
       JMP APPEND           ; REMOUT LINE
OPERTN
       JMP PRINTA
CLOSE
       LDA MNCFLG           ; MON IN EFFECT ?
       BEQ CLOSA            ; NO BRANCH
       JSR MOVE
       BEQ CLOSAA
       CMP #':
       BEQ CLOSAA
       BNE CLOSE
CLOSAA
       LDA #6
       STA AMOUNT
       JSR ADD
       LDA #':
       STA (TEMP),Y
       JSR MOVE
       LDA TEMP
       STA TMP
       LDA TEMP+1
       STA TMP+1
CLOSA
       LDA #$A0             ; CLOSE TOKEN
       LDY #0
       STA (TMP),Y
       LDA #$20             ; SPACE
       INY
       STA (TMP),Y
       LDA #'1              ; FILE NUMBER 14
       INY
       STA (TMP),Y
       LDA #'4
       INY
       STA (TMP),Y
       LDA #17
       INY
       STA (TMP),Y
CLOSEA
       INY
       LDA (TMP),Y          ; STORE SPACES TO END
       BEQ CLOSEB           ; OF LINE
       CMP #':              ; END OF LINE ?
       BEQ CLOSEB
       LDA #$20
```

```
        STA (TMP),Y
        BNE CLOSEA
CLOSEB
        JMP PRINTA
READH
        JMP READI
DELETA
        LDA DELSTG,X        ; POINT TO DEL LINE
        JMP DELETB
READ                        ; CLEAR WRITE & DELETE FLAG
        LDA #0
        STA WRTFLG
        STA DELFLG
WRITEC
        LDA MNCFLG          ; MON IN EFFECT ?
        BNE READH           ; YES BRANCH
        JSR MOVE            ; MOVE TO NEXT CHAR
        LDA #20             ; INSERT TWENTY SPACES
        STA AMOUNT
        JSR ADD             ; DO IT
WRITCA
        LDA #159            ; REM OPEN TOKEN
        STA (TMP),Y         ; STORE IN LINE
        LDX #0
        LDA DELFLG
        BEQ READA
        LDX #1
READA
        LDA DELFLG
        BNE DELETA
        LDA READST,X        ; PICKUP NEW LINE
DELETB
        BEQ READC           ; CHECK FOR END
        JSR AMOVE           ; UPDATE POINTER
        STA (TMP),Y         ; STORE CHAR
        INX                 ; UPDATE STRING POINTER
        BNE READA           ; DO AGAIN
READC
        LDX #0
        LDA MNCFLG          ; MON IN EFFECT ?
        BNE READD           ; YES BRANCH
        CLC                 ; UPDATE PROGRAM POINTER
        LDA TEMP
        ADC #17
        STA TEMP
        LDA TEMP+1
        ADC #0
        STA TEMP+1
READD
```

```
        JSR MOVE
        CMP #$20            ; SKIP SPACES
        BEQ READD
        CMP #'"             ; CHECK FOR "READ" FORM
        BEQ READF
READE
        JSR AMOVE           ; MOVE WRITE POINTER
        STA (TMP),Y         ; WRITE FILE NAME
        JSR MOVE            ; TO NEXT CHAR
        CMP #',             ; CHECK FOR "READ FILE
        BEQ READG           ; NAME " FORM
        CMP #'"             ; SKIP VARIABLES
        BEQ READG
        BNE READE
READG
        LDX #2
        LDA DELFLG          ; DELETE COMMAND ?
        BEQ READGA          ; NO BRANCH
        LDX #0
READGA
        BNE DELETC          ; YES BRANCH
        LDA WRTFLG          ; WRITE COMMAND
        BNE READGB          ; YES BRANCH
        LDA REEDST,X        ; GET READ STRING
        JMP READGC
DELETC
        LDA DLSTGA,X        ; GET DELETE STRING
        JMP READGC
READGB
        LDA WRSTRG,X        ; GET WRITE STRING
READGC
        BEQ READGD          ; END OF STRING ?
        JSR AMOVE           ; MOVE WRITE POINTER
        STA (TMP),Y         ; STORE STRING
        INX
        BNE READGA          ; DO AGAIN
READGD
        JSR MOVE            ; MOVE READ POINTER
        BNE REAGDA          ; LOOK FOR END OF LINE
        JSR AMOVE
        JMP READFD          ; WRITE SPACES TO EOL
REAGDA
        CMP #':             ; LOOK FOR END OF LINE
        BNE READGD
        JSR AMOVE
        JMP READFD          ; END OF LINE ROUTINE
READF
        JSR AMOVE           ; STORE QUOTE
        STA (TMP),Y
```

```
        JSR AMOVE
        LDA #$AA          ; WRITE + TOKEN
        STA (TMP),Y
        JSR MOVE
        CMP #';
        BEQ READFA
        CMP #',           ; LOOK FOR VARIABLES
        BEQ READFA
        BNE READDD
READFA
        JSR MOVE          ; MOVE TO VARIABLE
READDD
        BEQ READFB        ; NAME
        CMP #':           ; LOOK FOR END OF LINE
        BEQ READFB
        JSR AMOVE         ; READ AND STORE
        STA (TMP),Y       ; UNTIL END OF LINE
        JMP READFA
READFB
        JSR AMOVE
        LDX #0
        LDA DELFLG
        BEQ READFC
        LDX #1
READFC
        LDA DELFLG
        BNE DELETD
        LDA WRTFLG
        BNE WRITEA
        LDA REEDST,X      ; GET READ STRING
        JMP WRITEB
WRITEA
        LDA WRSTRG,X      ; GET WRITE STRING
WRITEB
        BEQ READFD        ; END OF STRING
        STA (TMP),Y       ; NO STORE STRING
        JSR AMOVE
        INX
        BNE READFC
DELETD
        LDA DLSTGA,X      ; GET DELETE STRING
        JMP WRITEB
READFD
        LDA MNCFLG        ; MON IN EFFECT ?
        BNE REAFDB        ; YES BRANCH
REAFDA
        LDA #$20          ; STORE SPACES TO
        STA (TMP),Y       ; END OF LINE
        JSR AMOVE
```

```
        LDA TEMP
        CMP TMP             ; END OF LINE ?
        BNE REAFDA          ; NO STORE AGAIN
        LDA TEMP+1
        CMP TMP+1
        BNE REAFDA          ; NO STORE AGAIN
        JMP PRINTA
REAFDB
REAFDC
        JSR MOVE
        BEQ REAFDD          ; SKIP INSERTED QUOTE
        CMP #':
        BEQ REAFDD
        BNE REAFDC
REAFDD
        JSR MOVE
        BEQ REAFDE
        CMP #':             ; SKIP TO INSERTED QUOTE
        BEQ REAFDE          ; OR END OF LINE
        BNE REAFDD
REAFDE
        LDA DELFLG
        BEQ REAFDA
        JSR MOVE
        BEQ REAFDA
        CMP #':             ; SKIP TO END OF LINE
        BEQ REAFDA
        BNE REAFDE
READI
        JSR AMOVE
        LDY #0
        LDA (TMP),Y
        BEQ READIA
        CMP #':             ; SKIP TO END OF LINE
        BEQ READIA          ; ROUTINE
        BNE READI
READIA
        LDA TEMP            ; SAVE READ POINTER
        PHA
        LDA TEMP+1
        PHA
        LDA TMP             ; SETUP POINTERS
        STA TEMP
        LDA TMP+1
        STA TEMP+1
        LDA #40             ; ADD SPACES
        STA AMOUNT
        JSR ADD
        PLA                 ; RETREIVE READ POINTER
```

```
        STA TEMP+1
        PLA
        STA TEMP
        LDY #0
        LDA #':              ; STORE COLON
        STA (TMP),Y
        JSR AMOVE
        JMP WRITCA           ; START WRITING STRING
AMOVE
        INC TMP              ; MOVES WRITE
        BNE AMOVE1           ; POINTER
        INC TMP+1
AMOVE1
        RTS
DELSTG
        .BYTE ' 15,8,15,"S0:',00
DLSTGA
        .BYTE '":',$A0,' 15',00
READST
        .BYTE ' 14,8,14,"0:',00
REEDST
        .BYTE $AA,'",S,R"',16,00
WRSTRG
        .BYTE $AA,'",S,W"',15,00
WRTFLG
        .BYTE 00
WRITE
        LDA #0               ; CLEAR DELETE FLAG
        STA DELFLG
        LDA #1
        STA WRTFLG           ; SET WRITE FLAG
        JMP WRITEC
APPEND
        LDA #143             ; REM TOKEN
        LDY #0               ; REMOUT LINE
        STA (LNSTRT),Y
        JMP PRINTA
MON
        JSR MOVE
        CMP #'C
        BEQ MONC
        CMP #'"
        BEQ MONEND
        BNE MON
MONC
        STA MNCFLG           ; SET MON FLAG
        BNE MON
MONEND
        JMP APPEND
```

```
MONRTN
        JMP PRINTA
MNCFLG
        .BYTE 00
NOMON
        JSR MOVE
        CMP #'C
        BEQ NOMONC
        CMP #'"
        BEQ NOMEND
        BNE NOMON
NOMONC
        LDA #0
        STA MNCFLG          ; CLEAR MON FLAG
        BEQ NOMON
NOMEND
        JMP APPEND          ; REM OUT LINE
NOMRTN
        JMP PRINTA
DELETE
        LDA #1
        STA DELFLG          ; SET DELETE FLAG
        LDA #0
        STA WRTFLG          ; CLEAR WRITE FLAG
        JMP WRITEC
DELFLG
        .BYTE 00
SAVK
        JMP SAVEK
RUNA
        LDA #147            ; RUN TOKEN
        BNE RUNB
SAVE
        LDA MNCFLG          ; MON IN EFFECT ?
        BNE SAVK
SAVEAA
        LDY #0
        LDA RUNFLG          ; RUN COMMAND ?
        BNE RUNA
        LDA #148            ; SAVE TOKEN
RUNB
        STA (TMP),Y
        JSR AMOVE
        LDA #$20            ; STORE SAVE AND SPACE
        STA (TMP),Y
        JSR AMOVE
        LDA #'"             ; STORE "
        STA (TMP),Y
SAVEA
```

```
        JSR MOVE
        CMP #$20            ; SKIP SPACES
        BEQ SAVEA
        CMP #'"             ; IS IT A QUOTE ?
        BEQ SAVEG           ; YES "SAVE" FORM
SAVEB
        JSR AMOVE           ; MOVE WRITE POINTER
        STA (TMP),Y         ; STORE FILENAME
        JSR MOVE            ; MOVE READ POINTER
        CMP #',             ; LOOK FOR START
        BNE SAVEFA          ; OF OPTIONS
SAVEC
        JSR MOVE
        CMP #'"             ; SKIP EVERYTHING TO
        BNE SAVEC           ; QUOTES
SAVED
        JSR AMOVE           ; WRITE ",8
        STA (TMP),Y
        JSR AMOVE
        LDA #',
        STA (TMP),Y
        JSR AMOVE
        LDA #'8
        STA (TMP),Y
        LDA MNCFLG          ; MON IN EFFECT ?
        BNE SAVEFB          ; YES BRANCH
SAVEE
        JSR MOVE            ; SKIP TO END OF LINE
        BEQ SAVEF
        CMP #':             ; END OF LINE ?
        BEQ SAVEF
        BNE SAVEE           ; NO MOVE AGAIN
SAVEF
        JSR AMOVE           ; MOVE WRITE POINTER
        LDA TMP
        CMP TEMP            ; AT END OF LINE ?
        BNE SAVEFC
        LDA TMP+1
        CMP TEMP+1
        BNE SAVEFC          ; NO STORE SPACE
        LDA #0              ; YES CLEAR RUN FLAG
        STA RUNFLG
        JMP PRINTA          ; DO AGAIN
SAVEFB
        LDA TMP             ; MAKE READ POINTER
        STA TEMP            ; EQUAL TO WRITE POINTER
        LDA TMP+1
        STA TEMP+1
        BNE SAVEE           ; ALWAYS
```

```
SAVEFC
        LDA #$20            ; WRITE SPACES
        STA (TMP),Y
        BNE SAVEF
SAVEFA
        CMP #'"             ; IS IT A QUOTE ?
        BNE SAVEB           ; NO IS PART OF FILENAME
        JMP SAVED           ; YES END OF FILENAME
SAVEG                       ; FILENAME IS A VARIABLE
        JSR MOVE
        CMP #$20            ; SKIP SPACES
        BEQ SAVEG
        CMP #';              ; LOOK FOR VARIABLE
        BEQ SAVEH           ; FILENAME
        CMP #',
        BEQ SAVEH
        BNE SAVEI
SAVEH
        JSR MOVE            ; SKIP , OR
        CMP #$20            ; SKIP SPACES
        BEQ SAVEH
SAVEI
        STA (TMP),Y         ; STORE VARIABLE
        JSR MOVE            ; FILENAME
        BEQ SAVEJ
        CMP #':             ; END OF LINE ?
        BEQ SAVEJ
        JSR AMOVE
        JMP SAVEI           ; NO DO AGAIN
SAVEJ
        LDA #$AA            ; END OF LINE ADD +
        JSR AMOVE           ; ADD ",8"
        STA (TMP),Y
        JSR AMOVE
        LDA #'"
        STA (TMP),Y
        JSR AMOVE
        LDA #',
        STA (TMP),Y
        JSR AMOVE
        LDA #'8
        STA (TMP),Y
        JSR AMOVE
        LDA #'"
        STA (TMP),Y
        LDA MNCFLG          ; MON IN EFFECT ?
        BNE SAVEJA          ; YES BRANCH
        JMP SAVEF           ; STORE SPACES TO END
SAVEJA
```

```
        JMP SAVEFB
SAVEK
        LDY #0              ; MON IN EFFECT
        JSR AMOVE           ; MOVE WRITE POINTER
        LDA (TMP),Y         ; TO END OF LINE
        BEQ SAVEKA          ; END BRANCH
        CMP #':
        BEQ SAVEKA
        BNE SAVEK           ; NO DO AGAIN
SAVEKA
        LDA TEMP            ; SAVE READ POINTER
        PHA
        LDA TEMP+1
        PHA
        LDA TMP             ; SETUP POINTERS
        STA TEMP
        LDA TMP+1
        STA TEMP+1
        LDA #40             ; ADD SPACES
        STA AMOUNT
        JSR ADD
        PLA                 ; RETREIVE READ POINTER
        STA TEMP+1
        PLA
        STA TEMP
        LDY #0
        LDA #':             ; WRITE COLON
        STA (TMP),Y
        JSR AMOVE
        JMP SAVEAA          ; GET FILENAME
RUN
        LDA #1
        STA RUNFLG          ; SET RUN FLAG
        JMP SAVE
RUNFLG
        .BYTE 00
CATALG
        LDA EMUFLG          ; EMULATE IN EFFECT ?
        BNE CATALD          ; YES BRANCH
        JMP APPEND          ; NO REMOUT LINE
CATALD
        LDA MNCFLG          ; MON IN EFFECT
        BEQ CATLGA          ; NO BRANCH
        LDY #0
        JSR MOVE            ; MOVE TO CHAR PASS CMD
        BEQ CATLGB          ; MOVE READ POINTER TO
        CMP #':             ; END OF LINE
        BEQ CATLGB
        BNE CATALD
```

```
CATLGB
        LDA TEMP            ; SETUP POINTERS TO NEW
        STA TMP             ; WRITE LOCATION
        LDA TEMP+1
        STA TMP+1
        LDA #3              ; ADD 3 SPACES
        STA AMOUNT
        JSR ADD
        LDA #':             ; ADD COLON
        STA (TMP),Y
        JSR AMOVE
CATLGA
        LDA #223            ; CATALOG TOKEN
        LDY #0
        STA (TMP),Y
        LDA TMP             ; SETUP POINTERS TO NEW
        STA TEMP            ; WRITE LOCATION
        LDA TMP+1
        STA TEMP+1
        JMP CLOSEA          ; MOVE TO END OF LINE
TOKCHG
        JSR START           ; TO BEGIN OF BASIC
        LDA #15             ; LOOK FOR WRITE FLAG
        STA SRHVAL
        JSR BEGIN
        LDA FOUND           ; FOUND IT ?
        BEQ DDDD            ; NO BRANCH CHECK READ
        LDA TMP             ; YES SAVE ADDRESS
        STA TMPST
        LDA TMP+1
        STA TMPST+1
        LDA #17             ; LOOK FOR CLOSE FLAG
        STA SRHVAL
        JSR BEGIN
        LDA FOUND           ; FOUND IT ?
        BEQ YYYY            ; NO USE PROGRAM END
        LDA TMP             ; YES SAVE CLOSE ADDRESS
        STA ENDST
        LDA TMP+1
        STA ENDST+1
ZZZZZ
        CLC                 ; START ADDRESS LESS THAN
        LDA TMPST+1         ; END ADDRESS ?
        CMP ENDST+1
        BCC ZZZA            ; YES LOOK FOR PRINT
        BEQ ZZZB            ; CHECK LS BYTE
        BCS DDDD            ; NO CHECK READ ADDRESS
ZZZB
        LDA TMPST           ; START ADDRESS LESS
```

```
        CMP ENDST+1          ; THAN END ADDRESS ?
        BCC ZZZA             ; YES LOOK FOR PRINT
        BCS DDDD             ; NO CHECK READ ADDRESS
BBB
        JMP BBBB
ZZZA
        LDA TMPST            ; GET WRITE ADDRESS
        STA TEMP
        LDA TMPST+1
        STA TEMP+1
        LDA #153             ; LOOK FOR PRINT TOKEN
        STA SRHVAL
        JSR BEGIN            ; DO IT
        LDA FOUND            ; FOUND IT ?
        BEQ BBB              ; NO BRANCH DELETE FLAGS
        LDA TMP+1            ; YES BETWEEN WRITE AND
        CMP ENDST+1          ; CLOSE
        BCC OKY              ; YES CHANGE TO PRINT#
        BEQ OKYA             ; CHECK LS BYTE
        BCS BBB              ; NO DELETE FLAGS
YYYY                         ; NO CLOSE LINE SAVE END OF
        LDA PGRMND           ; PROGRAM AS END
        STA ENDST            ; ADDRESS
        LDA PGRMND+1
        STA ENDST+1
        JMP ZZZZZ
DDDD
        JSR START            ; INITIALIZE TO START
        LDA #16              ; LOOK FOR READ FLAG
        STA SRHVAL
        JSR BEGIN
        LDA FOUND            ; FOUND IT ?
        BEQ QQQ              ; NO BRANCH
        LDA TMP              ; YES SAVE ADDRESS
        STA TMPST
        LDA TMP+1
        STA TMPST+1
        LDA #17              ; LOOK FOR CLOSE FLAG
        STA SRHVAL
        JSR BEGIN
        LDA FOUND            ; FOUND IT ?
        BEQ VVVV             ; NO USE PROGRAM END
        LDA TMP              ; YES SAVE ADDRESS
        STA ENDST            ; OF CLOSE
        LDA TMP+1
        STA ENDST+1
        JMP UUUU
QQQ
        JMP QQQQ
```

```
VVVV                           ; PROGRAM END AS END ADDRESS
        LDA PGRMND
        STA ENDST
        LDA PGRMND+1
        STA ENDST+1
        JMP UUUU
OKY
        LDY #0
        LDA #152               ; SAVE PRINT TO PRINT#14
        STA (TMP),Y
        JSR MOVE
        LDA #5
        STA AMOUNT
        JSR ADD
        LDA #'1
        STA (TEMP),Y
        JSR MOVE
        LDA #'4
        STA (TEMP),Y
        JSR MOVE
        LDA #',
        STA (TEMP),Y
        JMP TOKCHG
OKYA
        LDA TMP                ; PRINT BETWEEN WRITE AND
        CMP ENDST              ; CLOSE
        BCC OKY                ; YES CHANGE
BBBB                           ; NO DELETE FLAGS
        LDA TMPST              ; BY STORING SPACES
        STA TEMP               ; SETUP POINTERS
        LDA TMPST+1
        STA TEMP+1
        LDY #0
        LDA #$20               ; STORE SPACE
        STA (TEMP),Y
        LDA #17                ; FIND CLOSE FLAG
        STA SRHVAL
        JSR BEGIN
        LDA FOUND              ; FOUND IT ?
        BEQ BB                 ; NO BRANCH
        LDA #$20               ; DELETE FLAG
        STA (TMP),Y
BB
        JMP TOKCHG             ; TRY AGAIN
QQQQ
        JSR START              ; TO START OF PROGRAM
        LDA #17                ; LOOK FOR FLAG
        STA SRHVAL
        JSR BEGIN
```

```
        LDA FOUND           ; FOUND IT ?
        BNE TTTT            ; YES DELETE IT
        RTS                 ; NO END SUBROUTINE
TTTT                        ; DELETE FLAGS
        LDY #0
        LDA #$20
        STA (TMP),Y
        JMP TOKCHG          ; LOOK AGAIN
UUUU                        ; IS READ BEFORE CLOSE
        LDA TMPST+1
        CMP ENDST+1
        BCC PPPP            ; YES GOFOR IT
        BEQ OOOO            ; CHECK LS BYTE
        BCS QQQQ            ; NO DELETE CLOSE FLAG
OOOO
        LDA TMPST           ; LOWER ?
        CMP ENDST
        BCC PPPP            ; YES GOFOR IT
        BCS QQQQ            ; NO DELETE FLAG
PPPP
        LDA TMPST           ; GET READ ADDRESS
        STA TEMP
        LDA TMPST+1
        STA TEMP+1
        LDA #133            ; LOOK FOR INPUT TOKEN
        STA SRHVAL
        JSR BEGIN
        LDA FOUND           ; FOUND IT ?
        BEQ MMMM            ; NO BRANCH CHECK GET
        LDA TMP+1           ; YES BETWEEN READ AND
        CMP ENDST+1         ; CLOSE
        BCC OK              ; YES CHANGE
        BEQ OKA             ; CHECK LS BYTE
        BCS MMMM            ; NO CHECK GET
OKA
        LDA TMP             ; LS BYTE SMALLER ?
        CMP ENDST
        BCC OK              ; YES CHANGE
        BCS MMMM            ; NO CHECK GET
OK
        LDY #0
        LDA #132            ; CHANGE INPUT TOKEN
        STA (TMP),Y         ; TO INPUT# 14,
        JSR MOVE
        LDA #5              ; ADD ROOM TO ADD STRING
        STA AMOUNT
        JSR ADD
        LDA #'1             ; ADD 1
        STA (TEMP),Y
```

```
        JSR MOVE
        LDA #'4            ; ADD 4
        STA (TEMP),Y
        JSR MOVE
        LDA #',            ; ADD COMMA
        STA (TEMP),Y
        JMP TOKCHG         ; TRY AGAIN
MMMM
        LDA TMPST          ; SET UP POINTERS
        STA TEMP
        LDA TMPST+1
        STA TEMP+1
        LDA #214           ; CHECK FOR GET TOKEN
        STA SRHVAL
        JSR BEGIN
        LDA FOUND          ; FOUND IT ?
        BEQ LLLL           ; NO BRANCH DELETE FLAGS
        LDA TMP+1          ; YES BETWEEN READ AND
        CMP ENDST+1        ; CLOSE ?
        BCC OKAA           ; YES CHANGE GET TO GET#
        BEQ OKAAA          ; CHECK LS BYTE
        BCS LLLL           ; NO DELETE FLAGS
OKAAA
        LDA TMP            ; CHECK LS BYTE
        CMP ENDST
        BCC OKAA           ; LOWER CHANGE
        BCS LLLL           ; HIGHER DELETE FLAGS
OKAA                       ; CHANGE GIT TO GET# 14,
        LDA #161
        STA (TEMP),Y
        JSR MOVE
        LDA #5             ; MAKE ROOM
        STA AMOUNT
        JSR ADD            ; ADD # 14,
        LDA #'#
        STA (TEMP),Y
        JSR MOVE
        LDA #'1
        STA (TEMP),Y
        JSR MOVE
        LDA #'4
        STA (TEMP),Y
        JSR MOVE
        LDA #',
        STA (TEMP),Y
        JMP TOKCHG         ; TRY AGAIN
LLLL
        LDY #0             ; DELETE READ FLAG
        LDA TMPST
```

```
        STA TEMP
        LDA TMPST+1
        STA TEMP+1
        LDA #$20          ; REPLACE WITH SPACE
        STA (TEMP),Y
        LDA #17           ; CHECK CLOSE FLAG
        STA SRHVAL
        JSR BEGIN
        LDA FOUND         ; FOUND IT ?
        BEQ LL            ; NO SKIP
        LDA #$20          ; DELETE WITH SPACE
        STA (TEMP),Y
LL
        JMP TOKCHG
        .END
```

APPENDIX D

COMMODORE 64 - APPLE II EMULATION PROGRAM LISTINGS

```
; NAME OF THIS PROGRAM IS NEWCHARSET
        *=$9010             ; THIS PROGRAM
                            ; GENERATES NEW APPLE
                            ; CHARACTER SET AT $A000
       BSOUT=$FFD2
       BASIN=$FFCF
         TMP=$FB
        TEMP=$22
       LDA #147             ; YES CLEAR SCREEN
       JSR BSOUT
       LDA #142             ; TO UPPER CASE
       JSR BSOUT
       LDA #$7F             ; DISABLE INTERRUPTS
       STA $DC0D
       SEI
       LDA $00              ; SET DIRECTION TO OUTPUT
       ORA #$07
       STA $00
       LDA $01              ; DISABLE I/O AND EXPOSE
       AND #$FB             ; CHARACTER ROM
       STA $01
       LDA $D000            ; GET FIRST LOCATION
       STA $91A8            ; STORE FOR EMULATION
       LDY #0
       LDA #<53248          ; SETUP POINTER
       STA TMP              ; TO CHARACTER ROM
       LDA #>53248          ; FIRST UPPER CASE
       STA TMP+1
       LDA #<40960          ; SETUP POINTER TO
       STA TEMP             ; NEW CHARACTER SET
       LDA #>40960          ; LOCATION
       STA TEMP+1
       LDA #$D2             ; SET STOP LOCATION
       STA ENDLOC
       JSR CHRSET           ; TRANSFER CHARACTERS
       LDA #<55296          ; SETUP POINTER
       STA TMP              ; TO CHARACTER ROM
       LDA #>55296          ; SECOND LOWER CASE
       STA TMP+1
       LDA #<41472          ; SETUP POINTER TO
       STA TEMP             ; NEW CHARACTER SET
       LDA #>41472          ; LOCATION
       STA TEMP+1
       LDA #$DA             ; SET STOP LOCATION
       STA ENDLOC
       JSR CHRSET
       LDA #<54272          ; SETUP POINTER
       STA TMP              ; TO CHARACTER ROM
       LDA #>54272          ; REVERSE UPPER CASE
```

```
STA TMP+1
LDA #<41984        ; SETUP POINTER
STA TEMP           ; TO NEW CHARACTER SET
LDA #>41984        ; LOCATION
STA TEMP+1
LDA #$D6
STA ENDLOC
JSR CHRSET
LDA #<56320        ; SETUP POINTER
STA TMP            ; TO CHARACTER ROM
LDA #>56320        ; REVERSE LOWER CASE
STA TMP+1
LDA #<42496        ; SETUP POINTER
STA TEMP           ; TO NEW CHARACTER SET
LDA #>42496        ; LOCATION
STA TEMP+1
LDA #$DE
STA ENDLOC
JSR CHRSET
LDA #<53248        ; SETUP POINTER
STA TMP            ; TO CHARACTER ROM
LDA #>53248        ; COMMODORE CASE
STA TMP+1
LDA #<43008        ; SETUP POINTER
STA TEMP           ; TO NEW CHARACTER SET
LDA #>43008        ; LOCATION
STA TEMP+1
LDA #$D8
STA ENDLOC
JSR CHRSET
LDX #0             ; USED AS COUNTER SET
LDY #0
LDA #<40960        ; SETUP POINTER
STA TMP
LDA #>40960        ; MSB AT $A000
STA TMP+1
LDA #<41984        ; SETUP POINTER
STA TEMP
LDA #>41984        ; MSB AT $A000
STA TEMP+1
CLC
LDA TMP
ADC TABLE,X
STA TMP
INX
LDA TMP+1
ADC TABLE,X
STA TMP+1
DEX
```

```
        CLC
        LDA TEMP
        ADC TABLE,X
        STA TEMP
        INX
        LDA TEMP+1
        ADC TABLE,X
        STA TEMP+1
SETA    INX
        LDA TABLE,X
        CMP #$FF
        BEQ SETB
        STA (TMP),Y
        EOR #$FF
        STA (TEMP),Y
        INY
        CPY #8
        BNE SETA
        INX
        JMP SET
SETB    LDA #$1F            ; ENABLE CHRFLG
        STA $91A7           ; EMULATION PROGRAM
        LDA $00
        ORA #$07
        STA $00
        LDA $01             ; ENABLE I/O DISABLE
        ORA #$04            ; CHARACTER ROM
        STA $01
        LDA #129            ; ENABLE INTERRUPTS
        STA $DC0D
        LDA #147
        JSR BSOUT
        LDA #1              ; STORE ON SCREEN
        STA $8C01
        CLI
        RTS

TABLE
        .WORD 752
        .BYTE 18,42,68,0,0,0,0,0
        .WORD 760
        .BYTE 127,127,127,127,127,127,127,127
        .WORD 744
        .BYTE 48,8,12,2,12,8,48,0
        .WORD 736
        .BYTE 24,24,24,24,24,24,24,24
        .WORD 728
        .BYTE 12,16,48,64,48,16,12,0
        .WORD 224
```

```
        .BYTE 0,64,32,16,8,4,2,0
        .WORD 240
        .BYTE 0,16,40,68,0,0,0,0
        .WORD 248
        .BYTE 0,0,0,0,0,0,0,254
        .WORD 512
        .BYTE 128,64,32,0,0,0,0,0
        .WORD 0000
        .BYTE 255,255,255
CHRSET
        LDY #0
        LDA (TMP),Y         ; TRANSFER
        STA (TEMP),Y        ; CHARACTER SET
        INC TEMP
        BNE CHRSTA
        INC TEMP+1
CHRSTA  INC TMP
        BNE CHRSET
        INC TMP+1
        LDA TMP+1
        CMP ENDLOC
        BEQ CHRSTB
        BNE CHRSET
CHRSTB  RTS
ENDLOC  .BYTE 00
        .END
```

```
; PROGRAM IS NAMED BEGINA
; START OF EMULATION PROGRAM
; ZERO PAGE EQUATES
        ENDCHR=$08
         COUNT=$0B
        VALTYP=$0D
        GARBFL=$0F
        LINNUM=$14
         INDEX=$22
        FORPNT=$49
         JMPER=$54
         FACHO=$62
        FBUFPT=$71
        CHRGET=$73
        CHRGOT=$79
        TXTPTR=$7A
          TIME=$A0
           BYT=$A6
         ATMPA=$A8
        ATEMPA=$AA
           TMP=$FB
            AV=TMP
          TEMP=$FD
        XLACTE=$FD
; PAGE TWO EQUATES
;
           BUF=$200
        HIBASE=$288
        VECSAV=$2A7
;
; BASIC INDIRECT VECTORS
; PAGE THREE EQUATES
        IERROR=$300
         IMAIN=$302
        ICRNCH=$304
        IQPLOP=$306
         IGONE=$308
         IEVAL=$30A
         NMINV=$318
;
; BASIC ROM ROUTINES
;
        CHRFLG=$91A7
        CHRVAL=$91A8
         ERROR=$A437
          MAIN=$A483
         CRNCH=$A57C
         PRIT4=$A6EF
         PLOOP=$A6F3
```

```
         QPLOP=$A71A
        NEWSTT=$A7AE
          GONE=$A7E4
        PRINTC=$AAA0
         OUTDO=$AB47
        FRMNUM=$AD8A
        CHKNUM=$AD8D
          EVAL=$AE86
        PARCHK=$AEF1
        GETBYT=$B79E
        GETNUM=$B7EB
        GETADR=$B7F7
        FLOATC=$BC49
        SCRSCR=$E8EA
         GETIN=$F13E
         BASIN=$F157
         CHKIN=$F20E
         CLOSE=$F291
         CLRCH=$F333
          OPEN=$F34A
         BSOUT=$F1CA
        SETNAM=$FDF9
        SETLFS=$FE00
        BASOUT=$FFD2
        CHROUT=$FFD2
        CURSOR=$FFF0
;
; PROGRAM VARIABLES & CONSTANTS
;
        NEWTOK=$CC
        DATTOK=$49
        REMTOK=$55
;
        *=$91A9
EXGFLG
        .BYTE 00
MASK
        .BYTE 00
AREG
        .BYTE 00
XREG
        .BYTE 00
OUTFLG
        .BYTE 00
BLKFLG
        .BYTE 00
PRTFLG
        .BYTE 00
AMOUNT
```

```
        .BYTE 00
INTER
        .BYTE 00
LETTER
        .BYTE 00
BLINK
        .BYTE 00
RISE
        .BYTE 00
SLPFLG
        .BYTE 00
ENDROW
        .BYTE 00
SRTROW
        .BYTE 00
SMALLA
        .BYTE 00
BANK
        .BYTE 00
BSIGN
        .BYTE 00
ATMP
        .BYTE 00
ZROFLG
        .BYTE 00
SLPVAL
        .BYTE 00,00,00,00,00,00
BVALUE
        .BYTE 00,00,00,00,00,00
YLOCTE
        .BYTE 00
YCOORD
        .BYTE 00
ORFLAG
        .BYTE 00
NTRFLG
        .BYTE 00
CHAR
        .BYTE 00
IRQFLG
        .BYTE 00
GITFLG
        .BYTE 00
SMALL
        .BYTE 00
SLOW
        .BYTE 00
KEYFLG
        .BYTE 00
```

```
FLAG
        .BYTE 0
PDLNUM
        .BYTE 00            ; PADDLE NUMBER
COUNTA
        .BYTE 00
VALUE
        .BYTE 00
VALUEA
        .BYTE 00
COLFLG
        .BYTE 00
TOFLG
        .BYTE 00
EDCHRA
        .BYTE 00
FLSFLG
        .BYTE 00
NVRFLG
        .BYTE 00
LENGTH
        .BYTE 00
LENSTR
        .BYTE 00
SECOND
        .BYTE 00
DRVNUM
        .BYTE 00
NAME
        .BYTE 0,0,0,0,0,0,0,0,0,0,0
        .BYTE 0,0,0,0,0,0,0,0,0,0,0
        .BYTE 0,0,0,0,0,0,0,0,0,0,0
ADSFLG
        .BYTE 00
LENFLG
        .BYTE 00
SVEFLG
        .BYTE 00
STRVAL
        .BYTE 0,0,0,0,0
EXEFLG
        .BYTE 00            ; EXEC FLAG
;
EXFFLG
        .BYTE 00            ; EXEC FLG ACTIVE
STRLEN
        .BYTE 00
STRFLG
        .BYTE 00
```

```
YSTORE
        .BYTE 00
VALU    .WORD 0000
PLACE   .WORD 0000          ; HODS CURSOR LOC
        .WORD 0000
PLAIN   .WORD 0000
        .WORD 0000
        .BYTE 0
PLAINA
        .BYTE 00
        *=*+119
PLAINB
        .BYTE 0
        .WORD 0000
SYSFLG
        .BYTE 00            ; EMULATION FLAG
SRTADS  .WORD 0000
ENDADS  .WORD 0000
INHIBT  .WORD 0000
START   .WORD 0000
STRGND  .WORD 0000
BASLOC  .WORD 0000
LOCATE  .WORD 0000          ; HOLD BAS POINT
PDLONE  .WORD 0000          ; PADDLE VALUES
        .WORD 0000
CHRVEC  .WORD 0000
OUTVEC  .WORD 0000
MAINVC  .WORD 0000          ; WAITING LOOP
GETVEC  .WORD 0000
CLRVEC  .WORD 0000
STPVEC  .WORD 0000
LDEVEC  .WORD 0000
BRAKVC  .WORD 0000
IRQVEC  .WORD 0000
ERRSAV  .WORD 0000
RESVEC  .WORD 0000
RUN     .WORD 0000
ENDCOL  .WORD 0000
PDLKEY  .WORD 0000          ; PADDLE KEYS
SRTCOL  .WORD 0000
XCOORD  .WORD 0000
XLOCTE  .WORD 0000
OFFY    .WORD 0000
OFFX    .WORD 0000
TXTLOW  .WORD 0000
GRALOW  .WORD $2000
GRAHI   .WORD $4000
TEXTLO  .WORD $0400
SUMTMP
```

```
        .BYTE 0,0,0
SUMFLG
        .BYTE 0
ENDLOC
        .BYTE 0
CHKVAL
        .BYTE 0,0,0       ; START OF CHECK
FUDGE
        .BYTE 0           ; SUM CHECK POINT
HCLRZ
        .BYTE 0,80,64,16,0,128,96,16
ENAMEL  .WORD NAME
STVEC   .WORD HTAB-1
        .WORD HOME-1
        .WORD TRACE-1
        .WORD VTAB-1
        .WORD TEXT-1
        .WORD FLASH-1
        .WORD NVERSE-1
        .WORD HGR-1
        .WORD HPLOT-1
        .WORD POP-1
        .WORD GIT-1
        .WORD TROFF-1
        .WORD SPEED-1
        .WORD NRMAL-1
        .WORD LOMEM-1
        .WORD HIMEM-1
        .WORD HCOLR-1
        .WORD EXECUT-1
        .WORD BLODE-1
        .WORD CATALG-1
        .WORD PAUSE-1
        .WORD KILL-1
FUNVEC   .WORD PDL
ERRVEC   .WORD ERMSG0     ; ERROR MESSAGE
        .WORD ERMSG1      ; ADDRESS TABLE
        .WORD ERMSG2
        .WORD ERMSG3
        .WORD ERMSG4
        .WORD ERMSG5
        .WORD ERMSG6
        .WORD ERMSG7
        .WORD ERMSG8
;
IFADRS  .WORD IFTOKN-1
; INDIRECT VECTOR ADDRESSES
;
IVECS   .WORD TOKNIZ      ; TOKENIZATION
```

```
        .WORD PRTOK        ; PRINT TOKEN
        .WORD EXEST        ; EXEC STATEMENT
        .WORD EXEFUN       ; EXEC FUNCTION
;
PRNTBB  .WORD PRINTA-1
;
         FUNTOK=FUNVEC-STVEC/2+NEWTOK
;
MULTHI                     ; TABLE HIBYTES N*320

        .BYTE 0,1,2,3,5,6,7,8,10,11,12
        .BYTE 13,15,16,17,18,20,21,22,23
        .BYTE 25,26,27,28,30,31

MULTLO                     ; TABLE LOBYTES N*320

        .BYTE 0,64,$80,$C0,0,64,$80,$C0
        .BYTE 0,64,$80,$C0,0,64,$80,$C0
        .BYTE 0,64,$80,$C0,0,64,$80,$C0
        .BYTE 0,64
;
;
ERMSG0
        .BYTE 'ILLEGAL HCOLOR VALU',$C5
ERMSG1
        .BYTE 'ILLEGAL POSITION BYT',$C5
ERMSG2
        .BYTE 'ILLEGAL SPEED VALU',$C5
ERMSG3
        .BYTE 'ILLEGAL Y COORDINAT',$C5
ERMSG4
        .BYTE 'ILLEGAL X COORDINAT',$C5
ERMSG5
        .BYTE 'ILLEGAL HGR VALU',$C5
ERMSG6
        .BYTE 'FILE NAM',$C5
ERMSG7
        .BYTE 'HEX ADDRESS VALU',$C5
ERMSG8
        .BYTE 'PDL QUANITY VALU',$C5
;
; COMMAND WORDS TABLE
KEYTXT
        .BYTE 'HTA',$C2    ; HTAB
        .BYTE 'HOM',$C5    ; HOME
        .BYTE 'TRAC',$C5   ; TRACE
        .BYTE 'VTA',$C2    ; VTAB
        .BYTE 'TEX',$D4    ; TEXT
        .BYTE 'FLAS',$C8   ; FLASH
```

```
        .BYTE 'INVERS',$C5 ; INVERSE
        .BYTE 'HG',$D2     ; HGR
        .BYTE 'HPLO',$D4   ; HPLOT
        .BYTE 'PO',$D0     ; POP
        .BYTE 'GI',$D4     ; GIT
        .BYTE 'RAC',$C5    ; NOTRACE
        .BYTE 'SPEE',$C4   ; SPEED=
        .BYTE 'MA',$CC     ; NORMAL
        .BYTE 'LOME',$CD   ; LOMEM:
        .BYTE 'HIME',$CD   ; HIMEM:
        .BYTE 'HCO',$CC    ; HCOLOR=
        .BYTE 'EXE',$C3    ; 221
        .BYTE 'BLOD',$C5
        .BYTE 'CAT',$C1    ; CATALOG
        .BYTE 'PAUS',$C5   ; PAUSE
        .BYTE 'KIL',$CC    ; KILL
        .BYTE 'PD',$CC     ; PDL 226
        .BYTE 0            ; END OF TABLE
;
;
HPLOT
        CMP #$A4           ; IS IT A "TO" TOKEN
        BNE SIMPLE
        LDA #0
        STA SLPFLG         ; RESET MINUSSLOPE FLG
        JSR CHRGET         ; GET NEXT CHAR
        JSR GETCOR         ; GET X,Y COOR
        JSR ENDSTR         ; SAVE VALUE OF END
        JSR PLTLNE         ; PLOT LINE
        JMP CHRGOT         ; BACK TO BASIC
SIMPLE
        LDA #0
        STA SLPFLG         ; RESET MINUS SLOPE FLG
        JSR GETCOR         ; GET COORDINATES
        JSR STRSTR         ; STORE START LOCATION
        JSR CHRGOT         ; GET LAST CHAR
        CMP #$A4           ; IS IT A "TO" TOKEN
        BNE POINT          ; NO JUST POINT
EXTLNA
        JSR CHRGET         ; GET NEXT CHAR
        JSR GETCOR         ; GET END POINT
        JSR ENDSTR         ; SAVE END POINT
        JSR PLTLNE         ; PLOT LINE
        JSR CHRGOT         ; GET LAST CHAR
        CMP #$A4           ; CHECK FOR COMMA
        BEQ EXTLNA         ; GET NEXT LINE
        RTS                ; DONE BACK TO BASIC

POINT
```

```
        JSR PLOT           ; SET POINT
        JMP CHRGOT         ; BACK TO ROM STRSTR
                           ; SAVE START POINT
        LDA XCOORD         ; SAVE X COORDINATE
        STA SRTCOL         ; START COLUMN
        LDA XCOORD+1
        STA SRTCOL+1
        LDA YCOORD         ; SAVE Y COORDINATE
        STA SRTROW         ; START ROW
        RTS
PLTLNE                     ; PLOT LINE ROUTINE
        SEC                ; COMPARE COLUMN VALUES
        LDA ENDCOL         ; LOW BYTES
        SBC SRTCOL
        STA RUN            ; HIGH BYTES
        LDA ENDCOL+1
        SBC SRTCOL+1
        STA RUN+1
        ORA RUN
        BEQ AVRTLN         ; SRTCOL=ENDCOL
        BCC LFTLEE         ; ENDCOL<SRTCOL
        JMP RHTLNE         ; ENDCOL>SRTCOL
AVRTLN
        JMP VRTLNE
LFTLEE                     ; CORRECT MINUS RUN
        LDA RUN            ; GET ONES COMPLIMENT
        EOR #$FF
        STA RUN
        LDA RUN+1
        EOR #$FF
        STA RUN+1
        CLC                ; GET TWOS COMPLIMENT
        LDA RUN
        ADC #1
        STA RUN
        LDA RUN+1
        ADC #0
        STA RUN+1          ; TO LEFT LINE ROUTINE
        JMP LFTLNE
CONSTB                     ; GET Y INTERCEPT  B
        LDA SRTCOL+1       ; B=Y-MX:GET XCOORD
        LDY SRTCOL
        JSR $B395          ; GET XCOR TO FAC
        LDA #<SLPVAL       ; POINT TO MEMORY
        LDY #>SLPVAL       ; WHICH HOLDS SLOPE
        JSR $BA28          ; MOVE TO ARG
                           ; AND MULTIPLY
                           ; FAC=MX
        LDA $66            ; GET SIGN
```

```
        EOR SLPFLG
        STA $66             ; CORRECT SIGN
        JSR $BC0C           ; MOVE TO ARG
        LDY SRTROW          ; GET Y COOR
        JSR $B3A2           ; TO FP
        JSR $B853           ; AND SUBTRACT FAC=MX-Y
        LDA $66
        EOR #$80            ; INVERT SIGN
        STA $66
        STA BSIGN           ; FAC=Y-MX EQUALS CONSTANT B
        LDX #<BVALUE        ; POINT TO MEMORY
        LDY #>BVALUE
        JMP $BBD4           ; BVALUE =Y INTERCEPT B
SLOPE
                            ; GET MAGNITUDE OF SLOPE
        LDY RISE
        JSR $B3A2           ; CHANGE RISE TO FP
        JSR $BC0C           ; MOVE TO ARG
        LDA RUN+1           ; GET RUN
        LDY RUN
        JSR $B395           ; CHANGE RUN TO FP
        LDA $61
        JSR $BB12           ; SLOPE= RISE/RUN
SLOPEA
        LDX #<SLPVAL        ; POINT TO SLOPE
        LDY #>SLPVAL
        JMP $BBD4           ; STR SLOPE IN MEMORY
LFTLNE
                            ; ENDCOL<SRTCOL -RUN
        SEC                 ; CALCULATE RISE
        LDA ENDROW
        SBC SRTROW
        STA RISE
        BEQ HRALEE          ; HORIZONTAL LINE
        BCS LFTLAA          ; ENDROW>SRTROW +RISE
        EOR #$FF            ; -RUN AND -RISE
        ADC #1              ; GET TWOS COMPLIMENT
        STA RISE
        JMP LFTLNA          ; TO LEFT LINE ROUTINE
HRALEE                      ; TO HORIZONTAL LINE
        JMP HRALNE
LFTLAA
        LDA #$80            ; SET MINUS SLOPE
        STA SLPFLG          ; FLAG
LFTLNA                      ; PLOT LINE
        JSR SLOPE           ; GET SLOPE
        JSR CONSTB          ; GET Y INTERCEPT B
LFTLNB
        JSR PLOT            ; PLOT POINT
```

```
        DEC SRTCOL          ; GOTO NEXT X POINT
        LDA SRTCOL
        CMP #$FF
        BNE LFTLNC
        DEC SRTCOL+1
LFTLNC
        LDA SRTCOL+1        ; FIND Y COOR
        LDY SRTCOL
        JSR $B395           ; CONVRT TO FP
        LDA #<SLPVAL
        LDY #>SLPVAL
        JSR $BA28           ; MULTIPLY BY SLOPE
        LDA $66             ; CORRECT SIGN
        EOR SLPFLG
        STA $66
        LDA #<BVALUE        ; GET CONSTANT B
        LDY #>BVALUE
        JSR $BA8C
        LDA BSIGN           ; GET SIGN TO FAC
        STA $6E
        JSR $B86A           ; ADD CONSTANT B
        JSR $BC9B           ; CONVERT TO TWO BYTE
        LDY $65
        STY SRTROW          ; UPDATE Y COORDINATE
        JSR PLOT            ; PLOT POINT
        CLC                 ; CHECK FOR END OF LINE
        LDA SRTCOL
        CMP ENDCOL
        BNE LFTLNB          ; NO GET NEXT POINT
        LDA SRTCOL+1
        CMP ENDCOL+1
        BNE LFTLNB          ; NO GET NEXT POINT
        LDA ENDROW          ; YES PLOT END POINT
        STA SRTROW
        JMP PLOT            ; BACK TO BASIC
HRALNE                      ; ENDCOL<SRTCOL
        JSR PLOT            ; PLOT POINT
        DEC SRTCOL          ; GET NEXT X COORD
        LDA SRTCOL
        CMP #$FF
        BNE HRLNEB
        DEC SRTCOL+1
        JSR PLOT
HRLNEB
        LDA SRTCOL          ; END OF LINE
        CMP ENDCOL
        BNE HRALNE          ; NO DO AGAIN
        LDA SRTCOL+1        ; END OF LINE ?
        CMP ENDCOL+1
```

```
        BNE HRALNE          ; NO DO AGAIN
        JMP CHRGOT          ; YES DONE
RHTLNE                      ; ENDCOL>SRTCO +RUN
        SEC
        LDA ENDROW          ; GET RISE
        SBC SRTROW
        STA RISE
        BEQ HORLNE          ; HORIZONTAL LINE
        BCS RHTLNA          ; +RISE AND +SLOPE
                            ; -RISE AND - SLOPE
        EOR #$FF            ; TOGGLE BITS
        ADC #1              ; GET TWO COMPLEMENT
        STA RISE            ; SAVE RISE
        LDA #$80            ; SET SLPFLG
        STA SLPFLG
RHTLNA                      ; PLOT LINE
        JSR SLOPE           ; GET SLOPE
        JSR CONSTB          ; GET Y INTERCEPT B
RHTLNB
        JSR PLOT            ; PLOT POINT
        INC SRTCOL          ; GET NEXT X COORD
        BNE RHTLNC
        INC SRTCOL+1
RHTLNC
        LDA SRTCOL+1        ; POINT TO X
        LDY SRTCOL
        JSR $B395           ; CONVRT X COOR TO FP
        LDA #<SLPVAL        ; POINT TO SLOPE
        LDY #>SLPVAL
        JSR $BA28.          ; SLOPE TO ARG
                            ; MULTIPLY BY SLOPE
        LDA $66             ; GET SIGN
        EOR SLPFLG          ; AND CORRECT
        STA $66
        LDA #<BVALUE        ; POINT CONSTANT B
        LDY #>BVALUE
        JSR $BA8C           ; MULTIPLY TO FAC
        LDA BSIGN           ; GET SIGN
        STA $6E
        LDA $61
        JSR $B86A           ; ADD CONSTANT B
        JSR $BC9B           ; CONVERT TO TWO BYTE
        LDY $65
        STY SRTROW          ; SAVE NEW Y COOR
        JSR PLOT            ; PLOT POINT
        CLC                 ; CHECK FOR END OF LINE
        LDA SRTCOL          ; LSB
        CMP ENDCOL
        BNE RHTLNB          ; NO DO AGAIN
```

```
        CLC
        LDA SRTCOL+1        ; CHECK MSB
        CMP ENDCOL+1        ; END OF LINE
        BNE RHTLNB          ; NO DO AGAIN
        LDA ENDROW          ; YES PLOT END POINT
        STA SRTROW
        JMP PLOT            ; PLOT AND RETURN
HORLNE                      ; HORIZONTAL LINE ROUTINE
        JSR PLOT            ; PLOT POINT
        INC SRTCOL          ; GET NEXT POINT
        BNE HRLNEA
        INC SRTCOL+1
        JSR PLOT            ; PLOT POINT
HRLNEA
        LDA SRTCOL          ; CHECK FOR END
        CMP ENDCOL          ; OF LINE LSB
        BNE HORLNE          ; NO DO AGAIN
        LDA SRTCOL+1        ; CHECK MSB
        CMP ENDCOL+1
        BNE HORLNE          ; NO DO AGAIN
        JMP CHRGOT          ; YES RETURN
VRTLNE                      ; VERTICAL LINE ROUTINE
        SEC
        LDA SRTROW          ; CHECK ROW COOR
        SBC ENDROW
        BEQ PLOTD           ; SAME STRTAND ENDPOINT
        BCS UPLINE          ; SRTROW >ENDROW
VRTA
        JSR PLOT            ; SRTROW <ENDROW
        INC SRTROW          ; PLOT VERTICAL LINE
        JSR PLOT            ; PLOT POINT
        LDA SRTROW          ; CHECK END OF LINE
        CMP ENDROW
        BNE VRTA            ; NO DO AGAIN
        RTS                 ; YES DONE
PLOTD
        JMP PLOT
UPLINE                      ; VERTICAL LINE BOTTOM
        CLC                 ; TO TOP
        JSR PLOT            ; PLOT POINT
        DEC SRTROW          ; GET NEXT ROW COOR
        JSR PLOT            ; PLOT POINT
        LDA SRTROW          ; CHECK FOR END
        CMP ENDROW
        BNE UPLINE          ; NO DO AGAIN
        RTS                 ; YES DONE

ENDSTR                      ; SAVES END COOR VALUES
        LDA XCOORD          ; SAVE X COOR
```

```
        STA ENDCOL
        LDA XCOORD+1
        STA ENDCOL+1
        LDA YCOORD          ; SAVE Y COOR
        STA ENDROW
        RTS
GETCOR
        JSR GETNUM          ; XTO$14,YTOX-REG
        CLC
        CPX #200
        BCS HPLERY          ; CHECK Y COORDINATE
        LDA $15             ; CHECK X COORDINATE
        CMP #1
        BEQ OK
        CMP #00
        BNE HPLERX          ; X>THAN 512
        BEQ OKA
OK
        CLC
        LDA $14             ; MUST BE <320
        CMP #64
        BCS HPLERX          ; X>320
OKA
        STX YCOORD          ; SAVE Y VALUE
        LDA $15             ; SAVE X VALUE
        STA XCOORD+1
        LDA $14
        STA XCOORD
        RTS
HPLERY
        LDA #$03            ; OUTPUT ERROR
        JMP GETERR          ; MESSAGES
HPLERX
        LDA #$04
        JMP GETERR
;
; PLOT POINT ROUTINE
;
PLOT
        LDA SRTROW          ; Y COORDNTE TOACC
        LSR A
        LSR A
        LSR A
        STA YLOCTE          ; SAVE Y COORDOF COLOR
        TAY                 ; OFFY=320*INT(Y/8)+(YAND7)
        LDA MULTLO,Y
        STA OFFY
        LDA MULTHI,Y        ; TIMES 320
        STA OFFY+1
```

```
        LDA SRTROW          ; GET Y COORDINATE
        AND #$07
        CLC                 ; PLUS Y AND 7
        ADC OFFY
        STA OFFY
        CLC
        LDA SRTCOL+1        ; GET X COORDOF POINT
        BEQ PLOTA           ; LESS THAN 256
        SEC
PLOTA
        LDA SRTCOL
        ROR A               ; DIVIDE BY 2
        CLC                 ; CLEAR IF SET
        LSR A               ; DIVIDE BY 4
        LSR A
        STA XLOCTE          ; SAVE X COORDOF COLOR
        LDA #0
        STA XLOCTE+1        ; OFFX=8*INT(X/8)
        LDA SRTCOL          ; GET X COORDINATE
        AND #$F8
        STA OFFX
        CLC                 ; AV=GRALOW+OFFY+OFFX
        LDA GRALOW          ; GET LOBYTE START OF
        ADC OFFY            ; GRAPHICS ADD OFFY
        STA AV
        LDA GRALOW+1        ; GET HI BYTE
        ADC OFFY+1          ; ADD HIBYTE
        STA AV+1
        CLC
        LDA AV              ; ADD OFFX DO TO X COORD
        ADC OFFX
        STA AV
        LDA AV+1
        ADC SRTCOL+1        ; GET HIBYTE OF OFFX
        STA AV+1            ; MA=2^((7-(X AND 7)
        LDA SRTCOL          ; GET X COORDINATE
        AND #$07            ; AND DATA
        EOR #$07            ; COMPLIMENT BITS
        TAX
                            ; MA USED AS COUNTER
        LDA #1              ; SHIFT 1 X TIMES
PLOTB
        DEX
        BMI PLOTC
        ASL A               ; SHIFT BIT LEFT
        BNE PLOTB           ; ALWAYS BRANCH
PLOTC
        LDY #0
        STA ATMP            ; SAVE A
```

```
        LDA IRQFLG
        BEQ PLOTE
        SEI                  ; DISABLE INTERRUPTS
        LDA #00              ; DISABLE RASTER INTERRUP
        STA $D01A
        LDA $01              ; SWITCH SCREEN RAM IN
        AND #$FD             ; DISABLES KERNAL
        STA $01
PLOTE
        LDA ATMP             ; RESTORE A
        ORA (AV),Y           ; GET OLD DATA
        STA (AV),Y           ; SET NEW DATA BITS
        LDA IRQFLG
        BEQ PLOTF
        LDA $01              ; ENABLE KERNAL
        ORA #$02
        STA $01
        LDA #$81             ; INTERRUPT
        STA $D01A
PLOTF
        CLI
        LDY YLOCTE           ; GET Y COORD OF COLOR
SETCOL
        CPY #0               ; USED AS COUNTER
        BEQ PLTEND           ; ARE WE DONE
        CLC
        LDA XLOCTE           ; GET COLUMN
        ADC #40              ; ADD 1 ROW
        STA XLOCTE
        LDA XLOCTE+1
        ADC #00
        STA XLOCTE+1
PLTLBL
        DEY
        JMP SETCOL           ; DOIT UNTIL Y IS ZERO
PLTEND
        CLC
        LDA XLOCTE           ; GET OFFSET
        ADC TEXTLO           ; GET START OF TEXT
        STA XLACTE           ; WINDOW UPDATE POINTR
        LDA XLOCTE+1         ; DO HI BYTE
        ADC TEXTLO+1
        STA XLACTE+1
        LDA COLFLG           ; GET COLOR
        LDY #0
        STA (XLACTE),Y       ; SET COLOROF POINT
        RTS
;
;
```

```
BEEP                        ; SOUND BELL
        PHP
        PHA
        TXA
        PHA
        TYA
        PHA
        LDA TMP             ; USED IN ERROR
        PHA                 ; HANDLER
        LDA TMP+1           ; SAVE TMP LOCATION
        PHA                 ; INTERRUPTS MUST BE
                            ; ENABLED FOR THIS
                            ; ROUTINE TO WORK
        LDA #$00            ; SET POINTER TO SID
        LDX #$D4
        STA TMP
        STX TMP+1
        LDY #$00            ; CLEAR SID
BEEPA
        LDA #$00
        STA (TMP),Y
        TYA
        CMP #$17
        BEQ BEEPB           ; DONE BRANCH
        CLC
        ADC #$01            ; INC POINTER BY ONE
        TAY
        JMP BEEPA           ; DO AGAIN
BEEPB
        LDA #$0F            ; SET UP SID FOR BELL
        STA $D418
        LDA #$00
        STA $D405
        LDA #$F7
        STA $D406
        LDA #$11
        STA $D404
        LDA #$28
        STA $D401
        LDA #$00
        STA $D400
        LDA #$00            ; CLEAR JIFFY CLOCK
        STA $A0
        STA $A1
        STA $A2
BEEPC
        LDA $A2             ; WAIT .1667 SECONDS
        CMP #$0A
        BEQ BEEPD           ; BRANCH IF DONE
```

```
        JMP BEEPC           ; NO DO AGAIN
BEEPD
        LDA #$10            ; SHUT OFF SID
        STA $D404
        PLA                 ; RESTORE TMP
        STA TMP+1
        PLA
        STA TMP
        PLA
        TAY
        PLA
        TAX
        PLA
        PLP
        RTS
;
; REM RESTORE WEDGE
; THIS ROUTINE WILL CHECK INTERRUPT
; DURING RUN/STOP RESTORE SEQUENCE
; IF EVERYTHING IS OKAY THEN WILL
; INITIALIZE BASIC AND REINSTALL
; APPLE EMULATION PROGRAM
;
;
RWDG
        PHA                 ; SAVE REGISTERS
        TXA
        PHA
        TYA
        PHA
        LDA #<BRK2          ; DISABLE RECURSION
        LDX #>BRK2          ; RESET RESTORE VECTOR
        STA NMINV
        STX NMINV+1
        JSR STOP            ; CHECK ON STOP KEY
        BNE NOPRES          ; NOT PRESS BRANCH
NEWRSR
        JSR $FD15           ; INIT VECTORS
        JSR $FDA3           ; INIT SID/VIC REGS
        JSR $E518           ; RESET SCREEN
        JSR CHKSUM
        LDA SUMFLG
        BNE NEWRAA
        JSR SETDEF          ; SET DEFAULT COLORS
        JSR INSTAL          ; INSTALL ROUTINE
        JSR BEEP            ; SOUND BEEP
NEWRAA
        JMP ($A002)
NOPRES
```

```
        JSR BRKINT          ; RESTORE WEDGE POINTER
        CLI
        JSR BEEP
        JMP $FEBC
BRK2
        PHA                 ; SAVE REGISTERS
        TXA
        PHA
        TYA
        PHA
        JSR STOP            ; CHECK STOP KEY
        BNE BRK2A           ; NOT PRESSED
        JMP NEWRSR          ; NEW ROUTINE
BRK2A
        CLI
        JSR BEEP
        JMP $FEBC           ; RETURN TO BASIC
STOP
        STA AREG            ; PRESERVE REG
        STX XREG
        JSR $F6BC           ; SET UP FOR STOP KEY
        JSR $FFE1           ; READ STOP KEY
        PHP                 ; SAVE STATUS
        LDX XREG            ; RESTORE REGISTERS
        LDA AREG
        PLP                 ; RETREIVE STATUS
        RTS                 ; RETURN
SETDEF
        LDA TEMP
        PHA
        LDA TEMP+1
        PHA
        LDA #0
        STA $D020           ; SET BORDER COLOR
        LDA #0
        STA $D021           ; SET BACKGROUND
        LDA #1
        STA $286
        LDA #$00            ; SET POINTERS TO COLOR
        TAY
        STA TEMP            ; MEMORY
        LDA #$D8            ; SET START POINTERS
        STA TEMP+1
CHGTXA
        LDA #1              ; STORE COLOR MEMORY
CHGTXB
        STA (TEMP),Y        ; D800-DC00
        INC TEMP            ; INC POINTER
        BNE CHGTXB          ; DO 256 BYTES
```

```
        INC TEMP+1          ; INCREMENT AND CHECK
        LDA TEMP+1
        CMP #$DC            ; END OF MEMORY
        BNE CHGTXA          ; NO DO AGAIN
        PLA
        STA TEMP+1
        PLA
        STA TEMP
        RTS
;
;
ERRHND
                            ; ERROR HANDLER
        PHP
        STA $33C            ; SAVE REGISTERS
        STX $33D
        STY $33E
        TXA
        CMP #$1F
        BCS ERRORA          ; DISPLAY ERROR
        JSR BEEP            ; SOUND BELL
        LDA #0              ; CLEAR EXEC FLAGS
        STA EXEFLG
        STA EXFFLG
        STA EXGFLG
        LDA #5              ; CLOSE EXEC FILE
        JSR CLOSE
        JSR $FFE7           ; TO DEFAULT DEVICES
        JSR TEXT
ERRORA
        LDA $33C            ; RESTORE REGS
        LDX $33D
        LDY $33E
        PLP
        JMP (ERRSAV)
; TO BASIC
;
;
; CALCULATES CHECKSUM OF PROGRAM
CHKSUM
        CLC                 ; CHECK PROGRAM DATA
        LDA TEMP
        PHA
        LDA TEMP+1
        PHA
        LDY #0              ; ZERO COUNTER AND
        STY TEMP            ; POINTERS
        STY SUMTMP
        STY SUMTMP+1
```

```
        STY SUMTMP+2
        LDA #$C0          ; SET START POINTER
        STA TEMP+1        ; FOR UPPER PROGRAM
        LDA #$CC          ; SET END POINTER
        STA ENDLOC
        JSR SLOOP         ; DO CHECKSUM
        LDA #<CHKVAL      ; SET LOW START POINT
        STA TEMP          ; LOW BYTE
        LDA #>CHKVAL      ; HI BYTE
        STA TEMP+1
        LDA #$A0          ; SET END  AT A000
        STA ENDLOC
        JSR SLOOP
        JMP CHECKA
SLOOP
        CLC               ; THREE BYTE CHECKSUM
        LDY #0
        LDA (TEMP),Y
        ADC SUMTMP
        STA SUMTMP        ; SAVE FIRST BYTE
        LDA SUMTMP+1
        ADC #0            ; GET CARRY
        STA SUMTMP+1      ; SAVE SECOND BYTE
        LDA SUMTMP+2
        ADC #0            ; GET CARRY
        STA SUMTMP+2      ; SAVE THIRD BYTE
        INC TEMP          ; ADVANCE READ POINTER
        BNE SLOOP         ; DONE NO CONTINUE
        INC TEMP+1        ; DO NEXT BLOCK
        LDA TEMP+1        ; CHECK IF END
        CMP ENDLOC        ; DONE YET ?
        BNE SLOOP         ; NO DO AGAIN
        RTS               ; RETURN

CHECKA                    ; CHECK WITH STORED VALUE
        LDA SUMTMP        ; CHECK LOW BYTE
        CMP CHKVAL
        BNE CORUPT        ; NOT SAME END
        LDA SUMTMP+1      ; CHECK MIDDLE BYTE
        CMP CHKVAL+1
        BNE CORUPT        ; NOT SAME END
        LDA SUMTMP+2      ; CHECK HIGH BYTE
        CMP CHKVAL+2
        BNE CORUPT        ; NOT THE SAME END
        LDA #0            ; SET GOOD FLAG
        STA SUMFLG
        BEQ SUMDIS        ; TO END
CORUPT
        LDA #1            ; SET CORRUPT FLAG
```

```
        STA SUMFLG
SUMDIS
        CMP #0
        BEQ SUMEND
        LDX #0
SUMDSA
        LDA CORRPT,X
        BEQ SUMEND
        JSR CHROUT
        INX
        BNE SUMDSA
CORRPT
        .BYTE 147,'PROGRAM CORRUPT'
        .BYTE 'ED-EXECUTION ABORTED',13,0
SUMEND
        PLA
        STA TEMP+1
        PLA
        STA TEMP
        LDA SUMFLG
        RTS
;
;
STPKEY                          ; USED IN STOP KEY
        LDA $91
        CMP #$7F
        BNE STPRTN
        PHP
        JSR $FFCC
        STA $C6
        PHA
        LDA #0                  ; ROUTINE
        STA EXEFLG              ; CLEAR EXEC FLAGS
        STA EXFFLG              ; RESETS DEFAULT
        STA EXGFLG              ; DEVICES
        LDA #5                  ; CLOSE EXEC FILE
        JSR CLOSE
        PLA
        PLP
STPRTN
        RTS
GETINA
        PHP
        LDA $11                 ; GET INPUT FLAG
        CMP #$40                ; $40=GET, 0=DIRECT
        BNE GETINB              ; NOT GET,TO ROM
        LDA GITFLG              ; IS IT APPLE 'GIT' ?
        BEQ GETINB              ; NO TO ROM
GET                             ; YES GET AND WAIT FOR KEYPRES
```

```
        PLP
        JSR GETIN           ; GET CHARACTER
        PHP                 ; SAVE STATUS
        CMP #0              ; NO KEY PRESSED ?
        BEQ GET             ; NO TRY AGAIN
        PLP
        RTS                 ; RESTORE STATUS
GETINB
        PLP
        JMP GETIN           ; GET CHAR AND RETURN
;
; OUR NEW ERROR MESSAGE ROUTINE
; STARTS HERE
;
GETERR
        PHA                 ; SAVE ERROR NUMBER
        JSR TEXT            ; TO TEXT MODE
        PLA                 ; RESTORE POINTER
        ASL A               ; DISPLAY ERRORUSED AS INDEX
        TAX
        LDA ERRVEC,X        ; POINT TO ERR TABLE
        STA INDEX           ; SET UP TEXT POINTER
        LDA ERRVEC+1,X
        STA INDEX+1
        JMP ERROR+16        ; DISPLAY ERROR
                            ; MESSAGE
;
;
PAUSE
        JSR FRMNUM          ; GET PAUSE VALUE
        JSR GETADR          ; NUMBER OF JIFFIES
        TAX                 ; TO INTEGER USED AS
PAUSE1                      ; COUNTER
        CPY #0              ; LOW BYTE ZERO ?
        BEQ PAUSE4          ; CHECK HIGH BYTE
PAUSE2
        DEY
        LDA TIME+2          ; SOFTWARE CLOCK
PAUSE3
        CMP TIME+2          ; ON SAME JIFFY
        BEQ PAUSE3          ; YES TRY AGAIN
        BNE PAUSE1          ; NO ONE JIFFY PASSED
PAUSE4
        CPX #$00            ; HI BYTE DONE
        BEQ PAUSE5          ; YES EXIT
        DEX                 ; NO COUNT DOWN
        JMP PAUSE2          ; AND DO AGAIN
PAUSE5                      ; DONE
        RTS
```

```
;
CLEAR
        LDY #0              ; CLEAR SCREEN
        LDA GRALOW+1        ; SETUP COUNTER
        STY TMP
        STA TMP+1
CLR1
        TYA
CLR2
        STA (TMP),Y         ; CLEAR BYTE
        INY
        BNE CLR2
        INC TMP+1
        LDA TMP+1
        CMP GRAHI+1         ; AT END OF SCREEN ?
        BNE CLR1            ; NO DO AGAIN
        RTS
;
HOME
        LDA #147            ; CLEAR SCREEN
        JSR BASOUT          ; HOME CURSOR
        NOP
        JMP CHRGOT
TRACE
        INC FLAG            ; SET TRACE FLAG ON
        BEQ TRACE           ; BESURE NOT ZERO
        JMP CHRGOT
TROFF
        LDA #0
        STA FLAG            ; SET TRACE FLAG OFF
        JMP CHRGOT
NRMAL
        CMP #176            ; LOOK FOR OR TOKEN
        BNE NRMALA          ; NO SYNTAX ERROR
        JSR CHRGET          ; GET NEXT CHAR
        CMP #217            ; MAL TOKEN
        BNE NRMALA          ; NO SYNTAX ERROR
        JSR CHRGET          ; GET NEXT CHAR
        SEI
        LDA #0
        STA NVRFLG          ; CLEAR FLAGS
        STA FLSFLG
        CLI
        LDA #146
        JSR BASOUT          ; STOP INVERSE MODE
        JMP CHRGOT
NRMALA
        JMP SYNTAX
;
```

```
;
FLASH
        SEI
        LDA #0              ; CLEAR INVERSE
        STA NVRFLG          ; MODE FLAG
        LDA #18
        STA FLSFLG          ; SET FLASH FLAG
        CLI
        LDA #146
        JSR BASOUT          ; STOP INVERSE MODE
        RTS
NVERSE
        SEI                 ; SET INVERSE MODE
        LDA #0
        STA FLSFLG          ; CLEAR FLASH FLAG
        LDA #18             ; SET INVERSE MODE
        STA NVRFLG          ; FLAG
        CLI
        JMP BASOUT
LOMEM                       ; SET LOMEM POINTERS
        CMP #':             ; COLON ?
        BEQ LOMEMA          ; YES CONTINUE
        JMP SYNTAX          ; NO SYNTAX ERROR
LOMEMA
        JSR CHRGET
        JSR FRMNUM          ; GET NEW LOCATION
        JSR GETADR          ; CONVRT FP TO INT
        STA $2E             ; START OF VARIABLE PNTER
        STY $2D             ; START OF ARRAYS
        STA $30             ; END OF ARRAYS
        STA $32
        STY $2F
        STY $31
        RTS
CLRALL                      ; CLEAR ALL FILES
        PHP                 ; SAVE REGS
        PHA
        LDA EXFFLG          ; IN EXEC MODE
        BNE CLRALA          ; YES BRANCH
        PLA                 ; RESTORE REGISTERS
        PLP
        JMP (CLRVEC)        ; NORMAL CLEAR
CLRALA
        PLA                 ; RESTORE REGS
        PLP
        RTS
        .FIL BEGINB
        .END
```

```
                              ; PROGRAM FILE NAME IS BEGINB
                              ; THIS SECTION CONTAINS MOSTLY
                              ; FLASH COMMAND ROUTINES
                             ; REMEMBER BEGINA PREVIOUS SECTION
MAINA
        PHP
        PHA
        LDA EXEFLG
        BNE MAINB
        LDA EXFFLG
        BNE MAINB
        PLA
        PLP
        JMP MAIN
MAINB
        LDA EXFFLG
        BEQ MAINC
        PLA
        PLP
        JMP RTURNA
MAINC
        PLA
        PLP
        JMP EXECAA
CLEARA                        ; CLEAR FLASH POINTERS
        LDA TEMP
        PHA
        LDA TEMP+1
        PHA
        LDA #<PLAIN           ; SET POINTERS
        STA TEMP
        LDA #>PLAIN
        STA TEMP+1
        LDY #0

        TYA
ZERO
        STA (TEMP),Y
        INY
        CPY #125              ; DONE YET ?
        BNE ZERO              ; NO DO AGAIN
        LDA #0                ; RESET ZERO FLAG
        STA ZROFLG            ; NO POINTERS FOR FLASH
        PLA
        STA TEMP+1
        PLA
        STA TEMP
        RTS
IRQRPT                        ; INTERRUPT HANDLER FOR HGR
```

```
        LDA IRQFLG          ; HGR FLAG SET
        BEQ ENTERA          ; NO TO FLASH HANDLER
        LDA $D019           ; YES CLEAR INTERRUPT
                            ; RASTER INTERRUPT OCCURED
        STA $D019
        LDA $D012           ; UPPER OR LOWER IRQ
        CMP #217
        BEQ LOW             ; TO TEXT MODE
        JSR HIRES           ; SET HIRES  SCREEN
        LDA #217            ; SET NEW INTERRUPT LOC
        BNE IRQDON
LOW                         ; BOTTOM OF SCREEN SETUP
        JSR TEXTA           ; TO TEXT MODE
        LDA #250            ; SET UPPER INTERRUPT LOC
IRQDON
        STA $D012           ; SET NEXT IRRUPT LOC
        LDA $DC0D           ; CHECK TIMER
        AND #$01
        BNE ENTER           ; YES PROCESS IRQ
        INC BLINK           ; CHECK FLASH TIMER
        BEQ CHANGE          ; CHANGE FLASH CODES
        JMP $FEBC           ; NO NOT YET RETURN
ENTER
        JMP (IRQVEC)
ENTERA
        INC BLINK
        BEQ CHANGE
        JMP (IRQVEC)
CHANGE
        LDA BLKFLG
        BNE CHNGEA
        LDA #1
        STA BLKFLG
        BNE CHNGEB
CHNGEA
        LDA #0
        STA BLKFLG
CHNGEB
        LDA ZROFLG
        BNE CHNGED
        JMP CHNGEC
CHNGED
        LDA TXTLOW
        STA ATEMPA
        LDA TXTLOW+1
        STA ATEMPA+1
        LDA #<PLAIN
        STA ATMPA
        LDA #>PLAIN
```

```
        STA ATMPA+1
        LDY #0
SCRPTA
        LDA (ATMPA),Y       ; CHECK FLASH PNTERS
        BEQ ANEXT
        LDX #8
SCRPTB
        ASL A
        BCC SCRPTC
        STA INTER
        LDA (ATEMPA),Y      ; GET SCREEN CHAR
        EOR #$80
        STA (ATEMPA),Y      ; STORE INVERSE
        LDA INTER
SCRPTC
        DEX
        INC ATEMPA
        BNE SCRPTD

        INC ATEMPA+1
SCRPTD
        CMP #0
        BEQ SCRPTE
        CPX #0
        BNE SCRPTB
        BEQ ANEXTA
SCRPTE
        TXA
        JMP AEXTAA
ANEXT
        LDA #8
AEXTAA
        STA AMOUNT
        CLC
        LDA ATEMPA
        ADC AMOUNT
        STA ATEMPA
        BCC ANEXTA
        INC ATEMPA+1
ANEXTA
        CLC
        LDA ATMPA
        ADC #1
        STA ATMPA
        BCC ANEXTB
        INC ATMPA+1
ANEXTB
        CMP #<PLAINB
        BNE SCRPTA
```

```
CHNGEC
        LDA #235
        STA BLINK
        LDA IRQFLG
        BEQ ENTERC
        JMP $FEBC
ENTERC
        JMP (IRQVEC)
INPUT                          ; SEE F157
        LDA $99
        BNE PUTA
        LDA $D3
        STA $CA
        LDA $D6
        STA $C9
        JMP INPUTC
PUTA
        CMP #$03               ; SEE F166
        BNE PUTB
        STA $D0
        LDA $D5
        STA $C8
        JMP INPUTC
PUTB
        JMP $F173
INPUTC                         ; SEE E632
        TYA
        PHA
        TXA
        PHA
        LDA $D0
        BEQ WAITA
        BNE INPUTE
WAIT                           ; SEE E5CA
        SEI
        JSR COMPAR
        JSR $E716
        CLI
WAITA                          ; SEE E5CD
        LDA $C6
        STA $CC
        STA $0292
        BEQ WAITA
        SEI
        LDA $CF
        BEQ WAITB
        LDA $CE
        LDX $0287
        LDY #$00
```

```
        STY $CF
        JSR $EA13
WAITB
        JSR $E5B4          ; SEE E5E7
        CMP #$83
        BNE WAITC
        LDX #$09
        SEI
        STX $C6
WAITBB
        LDA $ECE6,X        ; SEE E5F3
        STA $0276,X
        DEX
        BNE WAITBB
        BEQ WAITA
WAITC
        CMP #$0D           ; SEE E5FE
        BNE WAIT
        LDY $D5
        STY $D0
WAITCC
        LDA ($D1),Y        ; SEE E606
        CMP #$20
        BNE WAITD
        DEY
        BNE WAITCC
WAITD
        INY                ; SEE E60F
        STY $C8
        LDY #$00
        STY $0292
        STY $D3
        STY $D4
        LDA $C9
        BMI INPUTE         ; SEE E61D
        LDX $D6
        JSR $E6ED
        CPX $C9
        BNE INPUTE
        LDA $CA
        STA $D3
        CMP $C8
        BCC INPUTE
        BCS INPUTI
INPUTE                     ; SEE E63A
        LDY $D3
        LDA ($D1),Y
        STA $D7
        AND #$3F
```

```
        ASL $D7
        BIT $D7
        BPL PUTE
        ORA #$80
PUTE
        BCC PUTEA
        LDX $D4
        BNE PUTEB
PUTEA
        BVS PUTEB
        ORA #$40
PUTEB
        INC $D3
        JSR $E684
        CPY $C8
        BNE INPUTJ
INPUTI
        LDA #$00
        STA $D0
        LDA #$0D
        LDX $99
        CPX #$03
        BEQ WRITE
        LDX $9A
        CPX #$03
        BEQ WRITEB
WRITE
        SEI
        JSR COMPAR
        JSR $E716
        CLI
WRITEB
        JMP $E672
INPUTJ
        JMP $E674
OUTPUT
        PHA
        LDA $9A
        CMP #$03
        BNE OUTPTB
        CLI
OUTPTA
        LDA BLKFLG
        BNE OUTPTC
        SEI
        PLA
        JSR COMPAR
        JMP $E716
OUTPTB
```

```
        JMP $F1D5
OUTPTC
        LDA #$FF
        STA BLINK
        BNE OUTPTA
COMPAR
        PHP
        PHA                     ; SAVE REGISTERS
        STA LETTER
        TYA
        PHA
        TXA
        PHA
        LDA LETTER
        JSR WHERE
        LDA LETTER
        SEC
        JSR $E50A               ; GET CURSOR POS
        CMP #17                 ; DOWN KEY
        BEQ DOWN
        CMP #13                 ; LINE FEED
        BEQ DOWN
        CMP #147                ; CLR KEY
        BEQ ACLEAR
ROW     CPX #24                 ; BOTTOM ROW ?
        BNE AGET
        CPY #39                 ; RIGHT COLUMN ?
        BNE AGET
SHIFT
        JSR SCROLL
        JMP AGET
ACLEAR
        JSR CLEARA              ; CLEAR POINTER ARRAY
        JMP AGET
DOWN
        CPX #24                 ; BOTTOM ROW ?
        BEQ SHIFT
AGET
        PLA
        TAX
        PLA
        TAY
GETA
        PLA
        PLP
        RTS
SCROLL                          ; SCROLL ONE LINE
        PHP                     ; SAVE REGISTERS
        LDA TMP
```

```
        PHA
        LDA TMP+1
        PHA
        LDA TEMP
        PHA
        LDA TEMP+1
        PHA                 ; MOVE DATA UP FIVE BYTES
                            ; CLEAR LAST FIVE BYTES
        LDA ZROFLG          ; NO POINTERS ?
        BEQ ADONED          ; YES SKIP SCROLL
        LDY #0              ; POINTERS SCROLL
        STY ZROFLG          ; RESET FLAG FOR NO
                            ; POINTERS WILL SET FLAG IF ANY
        LDA #<PLAIN         ; SET POINTER TO BEG
        STA TMP             ; WRITE POINTER
        LDA #<PLAINA        ; READ POINTER
        STA TEMP
        LDA #>PLAIN
        STA TMP+1
        LDA #>PLAINA
        STA TEMP+1
LOOP
        LDA (TEMP),Y        ; GETLOWER LINE
        BEQ LOOP1B
        STA ZROFLG
LOOP1B
        STA (TMP),Y         ; PUTHIGH LINE
        INY
        CPY #125            ; END OF SCREEN ?
        BNE LOOP            ; NO DO AGAIN
        LDA #0              ; CLEAR LAST LINE
LOOP1
        STA (TMP),Y         ; STORE LINE
        INY
        BNE LOOP1           ; NO DO AGAIN
ADONED
        PLA
        STA TEMP+1
        PLA
        STA TEMP
        PLA
        STA TMP+1
        PLA
        STA TMP
        PLP
        RTS
;
ADANEA
        JMP ADONEA
```

```
WHERE                         ; CALCULATE CURSOR LOCATION
                              ; FOR FLASH MEMORY MODE
       LDA FLSFLG
       BEQ ADANEA
       LDA PRTFLG
       BEQ ADANEA
       SEC
       JSR $E50A              ; CURSOR POS TO X
       AND Y
       LDA #0
       STA BYT
       STA BYT+1
       CPX #0                 ; GET LINE LOCATION
       BEQ LOP
LOOP2
       CLC
       ADC #5
       DEX
       BNE LOOP2
LOP
       STA BYT                ; START OF CURSOR LINE
       TYA
       LSR A
       LSR A
       LSR A
       CLC
       ADC BYT
       STA BYT
       CLC
       LDA #<PLAIN
       ADC BYT
       STA BYT
       LDA #>PLAIN
       ADC BYT+1
       STA BYT+1
BEND
       TYA
       AND #7
       STA MASK
       SEC
       LDA #7
       SBC MASK
       STA MASK
       LDX MASK
       LDA #1
       CPX #0
       BEQ ADONE
LOOP4
       ASL A
```

```
        DEX
        CPX #0
        BNE LOOP4
ADONE
        STA MASK
        LDY #0
        SEC
        LDA LETTER
        CMP #5
        BEQ ADONEC
        CMP #28
        BEQ ADONEC
        CMP #30
        BEQ ADONEC
        CMP #31
        BEQ ADONEC
        CMP #32
        BCC ADONEA
        CMP #128
        BEQ ADONEA
        CMP #130
        BCC ADONEC
ADONEB
        CMP #160
        BCC ADONEA
ADONEC
        LDA #1              ; POINTERS ADDED
        STA ZROFLG          ; SET ZROFLG
        LDA (BYT),Y
        ORA MASK
        STA (BYT),Y
ADONEA
        RTS
                            ; INSTALL NEW INDIRECT VECTORS
                            ; A SYS TO INSTAL ACTIVATES OUR
                            ; NEW KEYWORD COMMANDS
                            ;
INSTAL
        SEI
        JSR CHKSUM
        LDA SUMFLG
        BEQ INSTAA
        RTS
INSTAA
        LDA #142            ; TO UPPER CASE
        JSR CHROUT
        LDA SYSFLG          ; EMULATION RUNNING
        BEQ INSTLA          ; NO BRANCH
        JSR KILL            ; DISABLE
```

```
INSTLA
        LDA #1              ; SET RUN FLG
        STA SYSFLG
        LDA $328
        STA STPVEC
        LDA $329
        STA STPVEC+1
        LDA #<STPKEY
        STA $328
        LDA #>STPKEY
        STA $329
        LDA $316
        STA BRAKVC
        LDA $317
        STA BRAKVC+1
        LDA #<NEWRSR
        STA $316
        LDA #>NEWRSR
        STA $317
        LDA $302
        STA MAINVC
        LDA $303
        STA MAINVC+1
        LDA #<MAINA
        STA $302
        LDA #>MAINA
        STA $303
        LDA $324            ; CHANGE VECTORS
        STA CHRVEC
        LDA $325
        STA CHRVEC+1
        LDA #<INPUT         ; CHANGE CHRIN VECTOR
        STA $324
        LDA #>INPUT
        STA $325
        LDA $314
        STA IRQVEC
        LDA $315
        STA IRQVEC+1
        LDA #<IRQRPT        ; CHANGE INTERRUPT
        STA $314            ; VECTOR
        LDA #>IRQRPT
        STA $315
        LDA IERROR
        STA ERRSAV
        LDA IERROR+1
        STA ERRSAV+1
        LDA NMINV
        STA RESVEC
```

```
        LDA NMINV+1
        STA RESVEC+1
        LDA #<ERRHND        ; SET ERROR ROUTINE
        STA IERROR          ; VECTOR
        LDA #>ERRHND
        STA IERROR+1
        JSR BRKINT
        LDA $326
        STA OUTVEC
        LDA $327
        STA OUTVEC+1
        LDA #<PRINT         ; CHANGE OUTPUTVECTOR
        STA $326
        LDA #>PRINT
        STA $327
        LDX #$07            ; FOUR TWO BYTE VECTORS
INSTL1
        LDA ICRNCH,X        ; SAVE OLD VECTORS
        STA VECSAV,X        ; TOKENIZATION
        LDA IVECS,X         ; PRINT TOKEN
        STA ICRNCH,X        ; EXECUTE STATEMENT
        DEX                 ; EXECUTE FUNCTION
        BPL INSTL1          ; KEEP GOING TILL DONE
INSTL2
        LDA #$FF            ; DEFAULTVALUE FOR SPEED
        STA SLOW
        STA $37             ; SET BASIC START OFSTRING
        STA $33             ; AND SET BASIC RAM END
        LDA #00             ; APPLE GIT FLAG
        STA TXTLOW
        STA GITFLG
        STA EXEFLG
        STA EXFFLG
        STA EXGFLG
        STA IRQFLG
        STA FLAG
        STA FLSFLG          ; FLASH FLAG
        STA COLFLG          ; BASIC DEFAULT CHRSET IF ZERO
        STA XLOCTE          ; HPLOT START INITIAL
        STA XLOCTE+1
        STA YLOCTE
        LDA $330
        STA LDEVEC
        LDA $331
        STA LDEVEC+1
        LDA #<ALOAD         ; LOAD VECTOR
        STA $330
        LDA #>ALOAD
        STA $331
```

```
        LDA $32A
        STA GETVEC
        LDA $32B
        STA GETVEC+1
        LDA #<GETINA        ; INPUT VECTOR
        STA $32A
        LDA #>GETINA
        STA $32B
        LDA $32C
        STA CLRVEC
        LDA $32D
        STA CLRVEC+1
        LDA #<CLRALL
        STA $32C
        LDA #>CLRALL
        STA $32D
        JSR CLEARA          ; CLEAR FLASH POINTERS
        LDA CHRFLG          ; CHARACTERSET ?
        CMP #$1F
        BNE INSTLB          ; NO BRANCH
        LDA CHRVAL          ; RESTORE FIRST DATA
        STA $A000
        LDA $D018           ; YES !!! SET VIC CHIP
        AND #$F0            ; TO CHARACTERSET LOC
        CLC
        ADC #$08
        STA $D018           ; TO A000
        LDA $DD02           ; SET TO OUTPUTS
        ORA #$03
        STA $DD02
        LDA $DD00           ; CHANGE TO BANK TWO
        AND #$FC
        ORA #$01
        STA $DD00
        LDA $D018           ; SET SCREEN TO $8C00
        AND #$0F
        ORA #$30
        STA $D018
        LDA #$8C            ; TELL KERNAL
        STA $288            ; SCREEN LOCATION
        LDA #147            ; CLEAR SCREEN
        JSR CHROUT
        LDA #$8B            ; SET HIGH RAM LOCATION
        BNE INSTLC
INSTLB
        LDA #$90
INSTLC
        STA $34
        STA $38
```

```
        LDA $288
        STA SMALLA
        STA TXTLOW+1
        LDA $DD00
        STA BANK
        LDA $D018           ; SCREEN MEMORY DEFAULT
        STA SMALL           ; CHAR MEMORY DEFAULT
        JSR SETDEF
        CLI
        JSR BEEP
        LDA #147
        JSR CHROUT
        CLC
        LDY #0
        LDA TXTLOW
        ADC #120
        STA TEMP
        LDA TXTLOW+1
        STA TEMP+1
INSTLD
        LDA TXTMSG,Y
        BEQ INSTLE
        STA (TEMP),Y
        INY
        BNE INSTLD
TXTMSG  .BYTE '                '
         .BYTE 1,16,16,12,5,0  ; APPLE
INSTLE
        JSR TEXT
        RTS                 ;
BRKINT
        LDA $DC0E           ; DISABLE INTERRUPTS
        AND #$FE
        STA $DC0E
        LDA #<RWDG          ; INITIALIZE WEDGE
        STA NMINV
        LDA #>RWDG
        STA NMINV+1
        LDA $DC0E           ; ENABLE INTERRUPTS
        ORA #$01
        STA $DC0E
        RTS

       *=$C000              ; PROGRAM CONTINUES
                           ; THIS SETS START ADDRESS THE SAME
                           ; SO THAT IT IS EASIER TO REMEMBER
        JMP INSTAL
        .FIL MIDDLE
```

```
; START OF PROGRAM MIDDLEA
; PATCH TO TOKENIZATION ROUTINE
; ALLOWS US TO TOKENIZE OUR OWN
; KEYWORDS USING THE UNUSED TOKEN
; NUMBERS 204-254
;
MOVEA
       JMP MOVE
SKQUTA
       JMP SKQUOT
TOKNIZ                        ; TOKENIZE ROUTINE
       JSR CRNCH              ; TOKENIZE AS USUAL
CRUNCH
                              ; DO SECOND TOKENIZATION
       LDX #$00               ; SET READ INDEX
       LDY #$04               ; SET WRITE INDEX
       STY GARBFL             ; CLEAR DATA FLAG
CRN1
       LDA BUF,X              ; GET NEXT VARIABLE
       BMI MOVEA
CRN2
       STA ENDCHR             ; FOR END QUOTE TEST
       CMP #$22               ; QUOTE
       BEQ SKQUTA             ; SKIP TO NEXT QUOTE
       BIT GARBFL             ; IF IN DATA STATEMENT
       BVS MOVE               ; WRITE THE CHARACTER
       CMP #'A                ; < THAN LETTER "A"
       BCC MOVE               ; YES WRITE IT
       CMP #$5B               ; > THAN LETTER Z
       BCS MOVE               ; YES WRITE IT
       STY FBUFPT             ; SAVE WRITE INDEX
       LDY #NEWTOK-$80        ; # OF 1ST TOKEN
       STY COUNT              ; SET TOKEN COUNTER
       LDY #$FF
       STX TXTPTR             ; SAVE READ INDEX
       DEX                    ; OFFSET INDEX
CRN3                          ; ADVANCE WRITE INDEX
       INY                    ; ADVANCE READ INDEX
       INX
CRN4
       LDA BUF,X              ; GET CHARACTER
       CMP #'B
       BNE CRN4A
       INX
       LDA BUF,X
       CMP #148
       BEQ CRN5A
       CMP #147
       BEQ CRN5A
```

```
        CMP #138
        BEQ CRN5A
        DEX
        LDA BUF,X
        JMP CRN4A
CRN5A
        DEX
        LDA #222            ; B TOKEN
        JMP CRN5
CRN5B
        DEX
        DEX
        DEX
        DEX
        LDA #217            ; N TOKEN
        JMP CRN5
CRN4A
        CMP #'N
        BNE CRN4C
        INX
        LDA BUF,X
        CMP #176
        BNE CRN4B
        INX
        LDA BUF,X
        CMP #'M
        BNE CRN4BA
        INX
        LDA BUF,X
        CMP #'A
        BNE CRN4BB
        INX
        LDA BUF,X
        CMP #'L
        BEQ CRN5B
        DEX
CRN4BB
        DEX
CRN4BA
        DEX
CRN4B
        DEX
        LDA BUF,X
CRN4C
        SEC
        SBC KEYTXT,Y        ; NEXT TABLE CHAR
        BEQ CRN3            ; YES KEEP GOING
        CMP #$80            ; LAST KEYWORD CHAR
        BNE NEXTKW          ; NO TRY NEXT WORD
```

```
        ORA COUNT           ; YES GET TOKEN NUMBER
CRN5
        LDY FBUFPT          ; RESTORE WRITE INDEX
MOVE
        INX                 ; ADVANCE READ INDEX
        INY                 ; ADVANCE WRITE INDEX
        STA BUF-5,Y         ; WRITE CHARACTER
        LDA BUF-5,Y         ; TEST FOR END OF LINE
        BEQ EXIT            ; YES END OF LINE
        SEC
        SBC #':             ; STATEMENT TERMINATOR ?
        BEQ MOVE1           ; YES CLEAR DATA FLAG
        CMP #DATTOK         ; TOKEN FOR DATA
        BNE MOVE2           ; DONT CLEAR FLAG
MOVE1
        STA GARBFL          ; CLEAR DATA FLAG
MOVE2
        SEC
        SBC #REMTOK         ; TOKEN FOR REM ?
        BNE CRN6            ; NO NEXT CHARACTER
        STA ENDCHR          ; YES FALL THRU
SKIP1
        LDA BUF,X           ; GET NEXT CHARACTER
        BEQ MOVE            ; KEEP GOING UNTIL EOL
        CMP ENDCHR          ; OR TERMINATOR
        BEQ MOVE
SKQUOT                      ; SKIP TEXT IN ""
        INY                 ; ADVANCE WRITE INDEX
        STA BUF-5,Y         ; WRITE CHAR
        INX                 ; ADVANCE READ INDEX
        BNE SKIP1           ; ALWAYS KEEP GOING
NEXTKW                      ; TRY NEXT KEYWORD
        LDX TXTPTR          ; RESTORE READ INDEX
        INC COUNT           ; ADVANCE KEYWORD CONTER
NEXT1
        INY                 ; ADVANCE TABLE INDEX
        LDA KEYTXT-1,Y      ; GET TABLE CHAR
        BPL NEXT1           ; SKIP TO NEXT WORD
        LDA KEYTXT,Y        ; GET 1ST CHAR
        BNE CRN7            ; TRY AGAIN
        LDA BUF,X           ; END OF TABLE
        BPL CRN5            ; ALWAYS
EXIT
        STA BUF-3,Y         ; GET END OF LINE
        LDA #$FF            ; RESTORE TXTPTR
        STA TXTPTR          ; TO START OF BUFF
        RTS
CRN6
        JMP CRN1
```

```
CRN7
        JMP CRN4
;
; THIS PATCH TO THE LIST ROUTINE
; ALLOWS US TO EXPAND OUR TOKENS
; BACK TO ASCII TEXT,SO THAT THEY
; LIST OUT CORRECTLY
;
PRTOK                       ; PRINT OUR NEW TOKEN
        BPL PRINT1          ; <128 NOT A TOKEN
        CMP #$FF            ; IS IT PI
        BEQ PRINT1          ; YES PRINT IT
        BIT GARBFL          ; ARE WE IN QUOTES
        BMI PRINT1          ; YES PRINT IT
        CMP #NEWTOK         ; IS IT A NEW TOKEN ?
        BCC OLDPR           ; NO USE OLD ROUTINE
        CMP #222
        BNE PRTOKA
        LDA #'B
        JMP PRINT1
PRTOKA
        CMP #217
        BNE PRTOKC
        LDA ORFLAG
        CMP #176
        BEQ PRTOKB
        LDA #'N
        JMP PRINT1
PRTOKB
        LDA #217
PRTOKC
        SEC
        SBC #NEWTOK-1       ; GET TOKEN NUMBER
        TAX                 ; USE AS INDEX
        STY FORPNT          ; SAVE STATEMENT INDEX
        LDY #$FF
PRTOK1
        DEX                 ; NEXT KEYWORD
        BEQ PRLOOP          ; THIS IS THE ONE
PRTOK2
        INY                 ; GET NEXT LETTER
        LDA KEYTXT,Y        ; IN KEYWORD
        BPL PRTOK2          ; END OF KEYWORD
        BMI PRTOK1          ; NO NEXT LETTER
;
PRLOOP
        INY                 ; GET NEXT LETTER
        LDA KEYTXT,Y        ; IN KEYWORD
        BMI PRINT2          ; END OF KEYWORD
```

```
        JSR OUTDO           ; NO PRINT CHAR
        BNE PRLOOP          ; AND REPEAT
;
PRINT1
        JMP PLOOP           ; PRINT ONE CHARACTER
PRINT2
        JMP PRIT4           ; PRINT LAST CHARACTER
OLDPR
        STA ORFLAG
        JMP QPLOP           ; USE OLD ROUTINE
;
; THIS PATCH TO THE STATEMENT
; EXECUTION ROUTINE ALLOWS US TO
; CHECK FOR OUR NEW STATEMENT
; TOKENS,AND TO EXECUTE THEM
;
PRTCMD
        JSR PTCMDA
        JMP NEWSTT
PTCMDA
        LDA EXGFLG
        BEQ PTCMDB
        PLA
        PLA
PTCMDB
        LDA PRNTBB+1
        PHA
        LDA PRNTBB
        PHA
        JMP CHRGET
PRINTA
        PHP
        PHA
        LDA #1
        STA PRTFLG
        PLA
        PLP
        JSR PRINTC
        PHP
        PHA
        LDA #0
        STA PRTFLG
        PLA
        PLP
        RTS
;
STRCE
        JMP STRACE
OLDEXA
```

```
        JMP OLDEXE
EXEST
        LDA FLAG            ; TRACE FLAG SET ?
        BNE STRCE           ; YES DISPLAY LINE NUM
BCK
        JSR CHRGET          ; GET NEXT CHAR
BCKA
        JSR CHRGOT          ; GET LAST CHAR
        CMP #168            ; IS IT A "NOT" TOKEN
        BEQ BCKC            ; YES BRANCH
BCKB
        CMP #139            ; CHECK "IF" TOKEN
        BEQ IFTOKA          ; GOTO NEW IF ROUTINE
        CMP #153            ; PRINT CMD ?
        BEQ PRTCMD          ; YES TO NEW PRINT
        CMP #NEWTOK         ; CHECK FOR NEW TOKEN
        BCC OLDEXA          ; NO BASIC EXECUTE
        JSR EXE1            ; NEW COMMAND EXECUTE
        JMP NEWSTT          ; BACK TO INTEPRETER
BCKC
        LDA $7A             ; SAVE POINTER
        PHA
        LDA $7B
        PHA
        JSR CHRGET          ; GET NEXT TOKEN
        CMP #213            ; IS IT "NOTRACE" TOKEN
        BEQ BCKD
        STA NTRFLG
        PLA
        STA $7B
        PLA
        STA $7A
        JSR CHRGOT
        JMP BCKB
BCKD
        STA NTRFLG
        PLA
        PLA
        JMP BCKB
IFTOKE
        LDA EXGFLG
        BEQ IFTOK1
        PLA
        PLA
IFTOK1
        LDA IFADRS+1        ; GET NEW IF
        PHA                 ; ADDRESS ON STACK
        LDA IFADRS
        PHA
```

```
        JMP $0073           ; GOTO IF ROUTINE
IFTOKA
        JSR IFTOKE          ; EXEC NEW IF
        JMP NEWSTT          ; IF ROUTINE THEN BACK
IFTOKN                      ; NEW IF ROUTINE
        JSR $AD9E           ; EVAL EXPRESSION
        JSR CHRGOT          ; GET LAST CHARACTER
        CMP #$89            ; GOTO CODE ?
        BEQ IFTKNA          ; BRANCH IF YES
        LDA #$A7
        JSR $AEFF           ; RESULT OF IF TERM
IFTKNA  LDA $61             ; EXPRESSION TRUE ?
        BNE IFTKNB
        JSR $A909           ; FIND OFFSET
        BEQ IFTKNC
IFTKNB
        JSR CHRGOT          ; GET PRESENT CHAR
        BCS IFTKND          ; BACK TO INTERPRETER
        JMP $A8A0           ; TO GOTO COMMAND
IFTKNC
        JMP $A8FB           ; ESP TO NEW LINE
IFTKND
        CMP #139
        BEQ IFTOKE
        CMP #NEWTOK
        BCC IFTKNF
        JMP EXE1
IFTKNF
        JSR CHRGOT
        JMP $A7ED
STRACE
        LDA $9D             ; DIRECT MODE
        BNE BCK2            ; YES DO NOTHING
        LDA #'[             ; NO PROG MODE DISPLAY
        JSR BASOUT
        JSR $BDC9           ; LINE NUMBER IN
        LDA #']             ; BRACKETS
        JSR BASOUT
BCK2
        JMP BCK
;
EXE1
        TAY
        LDA EXGFLG
        BEQ EXE2
        PLA
        PLA
EXE2
        SEC
```

```
        TYA
        SBC #NEWTOK       ; GET ADDRESS CODE
        ASL A
        TAY
        LDA STVEC+1,Y     ; GET ADDRESS OF
        PHA               ; COMMAND ROUTINE
        LDA STVEC,Y
        PHA
        JMP CHRGET
; EXEC ROUTINE
;
OLDEXE
        LDA EXGFLG
        BEQ OLDEX1
        JSR CHRGOT
        JMP $A7ED
OLDEX1                    ; EXEC OLD BASIC COMMAND
        JSR CHRGOT        ; GET LAST CHAR
        JMP GONE+3
;
; THIS PATCH TO THE EVALUATION
; ROUTINE ALLOWS US TO CHECK FOR
; OUR NEW FUNCTION KEYWORDS,AND
; TO EVALUATE THEM,LEAVING THE
; RESULT IN THE FLOATING POINT
; ACCUMULATOR
;
EXEFUN
        LDA #0
        STA VALTYP        ; SET TO NON STRING
        JSR CHRGET
        CMP #$FF          ; IS IT PI ?
        BEQ OLDFUN        ; YES TO OLD EVALUATE
        CMP #FUNTOK       ; IS IT A NEW FUNCTION ?
        BCC OLDFUN        ; NO DO OLD EVALUATE
                          ; GET TOKEN #
        SEC
        SBC #FUNTOK
        ASL A             ; USE AS INDEX
        PHA
        JSR CHRGET        ; GET NEXT CHAR
        JSR PARCHK        ; GET EXPRESSION IN ()
        PLA               ; RESTORE INDEX
        TAY
        LDA FUNVEC,Y
        STA JMPER+1
        LDA FUNVEC+1
        STA JMPER+2       ; FORM POINTER
        JSR JMPER         ; EVALUATE FUNCTION
```

```
        JMP CHKNUM          ; CHECK VARAIBLE TYPE
                            ; AND RETURN
OLDFUN
        JSR CHRGOT          ; GET LAST CHAR
        JMP EVAL+7          ; TO OLD ROUTINE
;
;
PRINT                       ; PRINT ROUTINE HANDLER
        PHP
        STA CHAR            ; SAVE CHARACTER
        LDA PRTFLG
        BEQ PRINTB
        LDA SLOW            ; SPEED FLAG SET ?
        CMP #$FF            ; ALSO VALUE OF SPEED
        BNE SPDSLW          ; BRANCH TO PRINT DELAY
BCK1
        LDA NVRFLG
        BEQ PRINTB
        LDA CHAR            ; RETRIEVE CHARACTER
        CMP #$0D            ; CHECK FOR RETURN
        BNE PRINTB          ; PRINT IT
        JSR OUTPUT
        PLP
        LDA #18             ; SET INVERSE
        JMP OUTPUT
PRINTB
        LDA CHAR
        CMP #7              ; CHECK FOR SOUND
        BNE PNTBBB
        JSR BEEP
        LDA CHAR
        PLP                 ; AND RETURN TO ROM
        RTS
PNTBBB
        LDA CHAR
        PLP
        JMP OUTPUT
SPDSLW                      ; SLOW DELAY
        TYA                 ; SAVE REGISTERS
        PHA
        TXA
        PHA
        LDA SLOW
        EOR #$FF            ; SWAP BITSUSEDAS COUNTER
        LSR A               ; DECREASE TIME BY A
        LSR A               ; FACTOR OF THIRTY-TWO
        LSR A
        LSR A
        LSR A
```

```
        TAY
        LDX #0
        JSR PAUSE1          ; DELAY OUTPUT
        PLA                 ; RESTORE REGISTERS
        TAX
        PLA
        TAY
        JMP BCK1            ; PRINT CHAR

ALOAD                       ; LOAD VECTOR WEDGE USED TO
        STA $93             ; GET START ADDRESS
        LDA #00             ; SET FLAG
        STA $90             ; CLEAR STATUS
        LDA $BA             ; DEVICE ADDRESS
        BNE ALOADB          ; NOT ZERO CONTINUE
ALOADA
        JMP $F713           ; ILLEGAL DEVICE
TAPE
        JMP $F533           ; TAPE LOAD ROUTINE
ALOADB
        CMP #$03            ; SCREEN
        BEQ ALOADA          ; YES ERROR
        BCC TAPE            ; TO TAPE
        LDY $B7             ; LENGTH OF FILE NAME
        BNE ALOADC          ; NOT ZERO OKAY
        JMP $F710           ; MISSING FILE NAME
ALOADC
        LDX $B9             ; SECONDARY ADDRESS
        JSR $F5AF           ; SEARCH FOR FILENAME
        LDA #$60            ; SECONDARY ADDRESS
        STA $B9
        JSR $F3D5           ; OPEN FILE
        LDA $BA             ; DEVICE NUMBER
        JSR $ED09           ; SEND TALK
        LDA $B9             ; SEND SECONDARY ADDRESS
        JSR $EDC7
        JSR $EE13           ; GET BYTE FROM IEC
        STA $AE             ; SAVE ADDRESS VALUE
        STA START           ; LSB OF START ADDRESS
        LDA $90             ; GET STATUS
        LSR A
        LSR A
        BCS ALOADE          ; TIME OUT THEN ERROR
        JSR $EE13           ; NO GET NEXT BYTE
        STA $AF             ; SAVE ADDRESS VALUE
        STA START+1         ; MSB OF START ADDRESS
        TXA
        BNE ALOADD          ; SECOND ADRSS NOT ZERO
        LDA $C3             ; GET NEW ADDRES
```

```
        STA $AE
        STA START          ; SAVE NEW ADDRESS LSB
        LDA $C4
        STA $AF
        STA START+1        ; SAVE NEW ADDRESS MSB
ALOADD
        JMP $F4F0          ; BACK TO ROM
ALOADE
        JMP $F530

; PDL(X) FUNCTIONS GETS PADDLE
; VALUES TO BASIC VARIABLE
PDL
        LDA LINNUM+1       ; SAVE LINENUMBER
        PHA
        LDA LINNUM
        PHA
        JSR GETADR         ; GET PADDLE NUMBER
        LDA LINNUM+1       ; CHECK FOR>255
        CMP #0
        BNE PDLERR         ; PADDLE NUMBER ERROR
        LDA LINNUM         ; GET LOW BYTE
        CMP #4             ; CHECK FOR <4
        BCS PDLERR         ; PADDLE NUMBER ERROR
        STA PDLNUM         ; SAVE PADDLE NUMBER
        JSR GETPDL         ; GET PADDLE VALUES
        LDX PDLNUM         ; USED AS INDEX
        LDA PDLONE,X       ; GET VALUE
        STA FACHO+1        ; TO FAC
        LDA #0
        STA FACHO          ; SET MSB TO ZERO
        PLA                ; RESTORE LINE NUMBER
        STA LINNUM
        PLA
        STA LINNUM+1
        LDX #$90           ; SET EXPONENTS
        SEC
        JMP FLOATC         ; CONVERT INT TO FP
GETPDL

        SEI                ; NO KEYBOARD INTERRUPTS
        LDA #$80           ; SET FOR PADDLE 0 1
        JSR PDLGET         ; GET VALUES
        STX PDLONE         ; SAVE PADDLE 0
        STY PDLONE+1       ; SAVE PADDLE 1
        LDA $DC00          ; GET KEY A FROM CIA 1

        AND #$0C           ; MASK BITS
        STA PDLKEY         ; SAVE KEY VALUE
```

```
        LDA #$40            ; PARAMETER FOR PDDLE2,3
        JSR PDLGET          ; GET PADDLE VALUE 2,3
        STX PDLONE+2        ; SAVE PADDLE VALUE 2
        STY PDLONE+3        ; SAVE PADDLE VALUE 3
        LDA $DC01           ; GET KEY B FROM CIA

        AND #$0C            ; MASK REQUIRED BITS
        STA PDLKEY+1        ; SAVE KEY VALUE 2
        LDA #$FF            ; ENABLE KEYBOARD
        STA $DC02           ; SET AS OUTPUTS

        CLI
        RTS
PDLGET
        STA $DC00           ; SELECT PADDLE SET
        ORA #$C0            ; MASK BITS
        STA $DC02           ; ON OUTPUT
        LDX #$00            ; STABLIZE DELAY TIME
DLYTME
        DEX                 ; ROUTINE
        BNE DLYTME
        LDX $D419           ; GET VALUES OF PADDLES
        LDY $D41A
        RTS
PDLERR                      ; ERROR MESSAGE NUMBER 8
        LDA #8
        JMP GETERR
;
; 'KILL' DISABLES THE NEW COMMANDS
;
KILL
        LDX #$07            ; RESTOREINDIRECT VECTORS
KILL1
        LDA VECSAV,X
        STA ICRNCH,X
        DEX
        BPL KILL1

        SEI
        LDA ERRSAV
        STA IERROR
        LDA ERRSAV+1
        STA IERROR+1
        LDA IRQVEC+1        ; RESET IRQ VECTOR
        STA $315
        LDA IRQVEC
        STA $314
        LDA BRAKVC
        STA $316
```

```
LDA BRAKVC+1
STA $317
LDA #<MAIN
STA $302
LDA #>MAIN
STA $303
LDA STPVEC
STA $328
LDA STPVEC+1
STA $329
LDA OUTVEC+1        ; RESET OUTPUT VECTOR
STA $327
LDA OUTVEC
STA $326
LDA LDEVEC          ; RESET LOAD VECTOR
STA $330
LDA LDEVEC+1
STA $331
LDA GETVEC          ; RESET GETIN VECTOR
STA $32A
LDA GETVEC+1
STA $32B
LDA CLRVEC
STA $32C
LDA CLRVEC+1
STA $32D
LDA CHRVEC          ; RESET CHRIN VECTOR
STA $324
LDA CHRVEC+1
STA $325
JSR TEXT            ; TO TEXT MODE
LDA #$7F            ; DISABLE INTERRUPTS
STA $DC0D
LDA #00
STA $D01A
LDA RESVEC
STA NMINV
LDA RESVEC+1
STA NMINV+1
CLC                 ; TO BASIC DEFAULT SCREEN
LDA $DD02
ORA #$03
STA $DD02
LDA $DD00
AND #$FC
ORA #$03
STA $DD00
LDA $D018
AND #$0F
```

```
        ORA #$10
        STA $D018
        LDA $D018
        AND #$F0
        ORA #$04
        STA $D018
        LDA #$04            ; TELL KERNAL
        STA $288
        LDA #0
        STA SYSFLG          ; RESTORED VECTORS FLAG
        LDA #$81
        STA $DC0D
        CLI
        LDA #5              ; CLOSE EXEC FLAG
        JSR CLOSE
        RTS
;
;
SPEED
        CMP #178            ; EQUAL FOLLOWS ?
        BEQ SPEEDA
        JMP SYNTAX
SPEEDA
        JSR CHRGET
        JSR FRMNUM          ; GET NEXT PARAMETER
        JSR GETADR          ; FAC TO INTEGER

        CLC
        CMP #00
        BNE SPDERR          ; ERROR IF >255
        STY SLOW            ; USED AS COUNTER&FLAG
        RTS                 ; SAVE SPEED VALUE
SPDERR                      ; SPEED ERROR OUTPUT
        LDA #$02
        JMP GETERR
HTAB
        JSR FRMNUM          ; GETNEW POSITION
        JSR GETADR          ; CONVRT FP TO INT
        CMP #00
        BNE POSERR          ; ERROR IF > 255
        STA COUNTA          ; INITIALIZE ROW COUNT
        TYA                 ; GET VALUE IN A
        SEC
        SBC #1
        CMP #40
        BCC LOWER           ; BRANCH LOWER THAN 40
AGAIN
        INC COUNTA          ; ROUTINE TO
        SEC                 ; COUNT MULTIPLES OF FORTY
```

```
        SBC #40
        CMP #40
        BCC LOWER           ; BRANCH WHEN LOWER
        BCS AGAIN           ; NO DOIT AGAIN
LOWER
        STA VALUE           ; SAVE REMAINDER
        SEC                 ; IT IS THE COLUMN NUM
        JSR CURSOR          ; GET CURRENT ROW/COL
        TXA                 ; GET ROW NUMBER INTO A
        CLC
        ADC COUNTA          ; ADD NUMBER OF ROWS
        STA COUNTA          ; SAVE USED AS COUNTER
DOIT
        CLC
        LDA COUNTA
        CMP #25
        BCC PSTION          ; BRANCH IF < 25
        DEC COUNTA          ; DECREASE COUNTER
        JSR SCRSCR          ; SCROLL SCREEN
        JMP DOIT            ; DO IT AGAIN
PSTION
        LDY VALUE           ; GET COL NUMBER
        LDX COUNTA          ; GET ROW NUMBER
        CLC
        JMP CURSOR          ; SET CURSOR
POSERR                      ; ERROR VALUE GREATER THAN
        LDA #$01            ; 255
        JMP GETERR
VTAB
        JSR FRMNUM          ; GETNEW POSITION
        JSR GETADR          ; CONVRT FP TO INT
        CLC
        CMP #0              ; GREATER THAN 255 ?
        BNE POSERR          ; IF YES ERROR
        TYA                 ; GET VALUE OF VTAB
        BEQ POSERR          ; ERROR IF ZERO
        CMP #25
        BCS POSERR          ; ERROR IF >25
        STA VALUEA          ; SAVE VTAB VALUE ROW
        DEC VALUEA
        SEC
        JSR CURSOR          ; GET CURSOR POSITION
        LDX VALUEA          ; GET ROW INFO
        CLC
        JMP CURSOR          ; POSITION CURSOR
TEXT
        SEI
        LDA #$00            ; DISABLE RASTER INTERUPT
        STA $D01A
```

```
        LDA #0              ; RESET INTERRUPT FLAG
        STA IRQFLG
        LDA #$81
        STA $DC0D
        LDX #24
        LDY #0
        CLC
        JSR $FFF0
        LDA SMALLA          ; TELL KERNAL
        STA $288
        JSR TEXTA
        CLI
        RTS
TEXTA
        LDA $DD02           ; BESURE BITS 0AND1
        ORA #$03            ; OF LOC DD00  BANK
        STA $DD02           ; ARE OUTPUTS CONTROL
        LDA BANK            ; SET TO DEFAULT BANK
        STA $DD00           ; BASIC DEFAULT SCREEN
        LDA $D011
        AND #$5F            ; BIT MAP OFF
        STA $D011           ; TEXT SCREEN ON
        LDA SMALL           ; SET SCREEN MEMORY
                            ; SET CHARACTER MEMORY
                            ; TO BASIC DEFAULT
        STA $D018           ; ACTUALLY 53248
        RTS
HIMEM                       ; HIMEM ROUTINE
        CMP #':             ; CHECK FOR COLON
        BEQ HIMEMA
        JMP SYNTAX          ; SYNTAX ERROR
HIMEMA
        JSR CHRGET
        JSR FRMNUM          ; GETNEW LOCATION
        JSR GETADR          ; CONVRT FP TO INT
        STA $34             ; BEGIN OF STRINGS
        STY $33
        STA $38             ; RAM END POINTER
        STY $37
        RTS
HCOLR
        CMP #176            ; LOOK FOR "OR" TOKEN
        BEQ HCOLRA
        JMP SYNTAX
HCOLRA
        JSR CHRGET
        CMP #178            ; LOOK FOR "=" TOKEN
        BEQ HCOLRB
        JMP SYNTAX
```

```
HCOLRB
        JSR CHRGET          ; GET NEXT CHAR
        JSR GETBYT          ; GET COLR VALUE
        CPX #8              ; IN X-REG
        BCS HCLERR          ; JMP ILLEGAL QUANITY
        LDA HCLRZ,X         ; GET APPLE II
        STA COLFLG          ; SAVE COLOR VALUE
        RTS
HCLERR
        LDA #0
        JMP GETERR
HGR                         ; BIT MAPPED GRAPHICS
        BEQ HGR1            ; EOL NOT SECOND SCREEN
        JSR GETBYT          ; GET BYTE VALUE
        CPX #$02
        BEQ SECPGA          ; SECOND PAGE ?
        LDA #$05
        JMP GETERR          ; HGR VALUE ERROR
SECPGA
        JMP SECPGE
HGR1
        SEI
        LDA #$E0            ; SETPOINTERFOR HPLOT
        STA GRALOW+1        ; START HIGH BYTE
        LDA #0              ; END HIGH BYTE
        STA GRAHI+1         ; SETPOINTERFOR HPLOT
        STA GRAHI
        STA GRALOW
        STA TEXTLO
        LDA #$CC            ; TELL KERNAL SCREEN LOC
        STA $288
        STA TEXTLO+1
        JSR CLEAR           ; CLEAR HIRES SCREEN
        JSR HOME            ; CLEAR TEXT SCREEN
        JSR HIRES           ; TO BIT MAPPED SCREEN
        LDA SMALLA          ; RESTORE POINTER
        STA $288
        LDX #24
        LDY #0
        CLC
        JSR $FFF0
        LDA #$7F
        STA $DC0D
        LDA #$81            ; ENABLE INTERRUPTS
        STA $D01A
        STA IRQFLG
        CLI
        RTS
HIRES
```

```
        LDA $DD02           ; BANK SELECT TO
        ORA #$03
        STA $DD02           ; OUTPUTS
        LDA $DD00           ; TO BANK THREE
        AND #252
        STA $DD00
        LDA $D011           ; SETBIT5 ONFORHIRES
        AND #$7F
        ORA #$20
        STA $D011           ; BIT MAPPED GRAPHICS
        LDA #$38            ; SET START
                            ; AT 8192 ($2000)+BANK $C000
                            ; SEE CHARACTER MEMORY
                            ; SET SCREEN MEMORY TO
        STA $D018           ; LOCATION $CC00
                            ; $C000 + $0C00
        RTS
SECPGE
        SEI
        LDA #$00
        STA $D01A           ; CLEAR RASTER INTRPT
        STA IRQFLG          ; CLEAR FLAG
        LDA #$81
        STA $DC0D
        LDA #$40            ; SET POINTER FOR HPLOT
        STA GRALOW+1        ; START HIGH BYTE
        LDA #$60            ; END HIGH BYTE
        STA GRAHI+1         ; SET POINTERFOR HPLOT
        LDA #0
        STA GRAHI
        STA GRALOW
        STA TEXTLO
        LDA #$60
        STA $288            ; TELL KERNAL SCREEN LOC
        STA TEXTLO+1
        JSR CLEAR           ; CLEAR BIT MAPPED SCR
        JSR HOME            ; CLEAR TEXT SCREEN
        LDA SMALLA          ; RESTORE SCRN POINTER
        STA $288
        LDA #$80            ; SET VALUES FOR SCREEN
                            ; SCREEN STRT AT$6000
        STA $D018           ; SEE SCREEN MEMORY
                            ; SET CHARACTER MEMORY
                            ; START AT $4000
        LDA $DD02           ; BE SURE OUTPUTS
        ORA #$03
        STA $DD02           ; AT DD00
        LDA $DD00           ; SWITCH TO BANK ONE
        AND #252
```

```
        ORA #$02            ; STARTS AT LOCATION
        STA $DD00           ; 4000 HEX
        LDA $D011           ; GOTO BIT MAP MODE
        AND #$7F
        ORA #$20
        STA $D011           ; SEE BIT MAP MODE
        CLI
        RTS
POP                         ; POP COMMAND
        BNE POPRTN          ; ERROR BACK TO BASIC
        LDA #$FF            ; CLEAR FOR NEXT
        STA $4A             ; VARIABLE POINTER
        JSR $A38A           ; FIND GOSUB IN STK
        TXS                 ; RESET STACK POINTER
        CMP #$8D            ; IS IS GOSUB CODE ?
        BEQ POPA            ; YES
        JMP $A8E0           ; PROGRAM ERROR
POPA
        PLA                 ; PULL CODE
        PLA                 ; PULL LINE NUMBER
        PLA
        PLA                 ; PULL EXEC STT POINTER-ESP
        PLA
        JSR $A8F8           ; GET NEXT STATEMENT
POPRTN
        RTS                 ; BACK TO INTERPRETER
GIT                         ; APPLE GET COMMAND
        LDA #1              ; SET APPLE GET FLAG
        STA GITFLG
        JSR CHRGOT          ; GET LAST CHAR
        JSR $AB7B           ; BASIC GET ROUTINE
        LDA #0              ; RESET GET FLAG
        STA GITFLG
        JMP CHRGOT          ; BACK TO ROM
        .FIL HIENDA

                            ; REMEMBER NAME OF PROGRAM HIENDA
CATALG                      ; DISPLAY DIRECTORY
         CMP #188           ; LOG TOKEN
         BEQ CATALA
         JMP SYNTAX
CATALA
         JSR CHRGET         ; GET NEXT CHAR
         LDA #'$            ; $ IS FILE NAME
         STA $FB            ; SAVE
         LDA #$FB           ; ADDRESS OF LOW BYTE
         STA $BB
         LDA #0
         STA $BC            ; HIGH BYTE OF FILE NAME
```

```
        LDA #1
        STA $B7         ; SET LENGTH OF FILE NAME
        LDA #8          ; SET DEVICE ADDRESS
        STA $BA
        LDA #$60        ; SET SECONDARY ADDRESS
        STA $B9
        JSR $F3D5       ; OPEN FILE WITH NAME
        LDA $BA
        JSR $FFB4       ; SEND TALK
        LDA $B9         ; SEND SEC ADDRESS
        JSR $FF96
        LDA #0
        STA $90         ; CLEAR STATUS
        LDY #3
CATAL1
        STY $FB         ; SKIP THREE BYTES
        JSR $FFA5       ; GET BYTE FROM FLOPPY
        STA $FC         ; SAVE IT
        LDY $90         ; STATUS OK ?
        BNE CATAL4      ; NO GET OUT
        JSR $FFA5       ; GET BYTE FROM FLOPPY
        LDY $90         ; STATUS OKAY ?
        BNE CATAL4      ; NO GET OUT
        LDY $FB         ; GET COUNTER
        DEY             ; DECREMENT
        BNE CATAL1      ; NOT DONE
        LDX $FC         ; OUTPUT NUMBER OF BLKS
        JSR $BDCD       ; USED
        LDA #$20        ; OUTPUT SPACE
        JSR $FFD2
CATAL3
        JSR $FFA5       ; GET NEXT BYTE
        LDX $90         ; GET STATUS
        BNE CATAL4      ; NO GET OUT
        TAX             ; ZERO ?
        BEQ CATAL2      ; END OF LINE
        JSR $FFD2       ; NO OUTPUT
        JMP CATAL3      ; GET NEXT CHAR
CATAL2
        LDA #13         ; OUTPUT "CR"
        JSR $FFD2
        LDY #2          ; TWO BYTE ADDRESS
        BNE CATAL1      ; AND CONTINUE
CATAL4
        JSR DISERR      ; DISPLAY ERROR
        JMP $F642       ; CLOSE FILE&DONE
SDONE
        JSR CHCKNM
        JSR CHRGOT
```

```
         JSR $B113
         BCS SDONEE
         CMP #',
         BNE SDONEA
         JSR CHRGET
         JMP SDONEE
SDONEA
         CMP #';
         BNE SDONEB
         JSR CHRGET
         JMP SDONEE
SDONEB
         CMP #':
         BNE SDONEC
         JMP DONEA
SDONEC
         CMP #0
         BNE SDONED
         JMP DONEA
SDONDD
         JMP SYNTAX
SDONED
         CMP #'"
         BNE SDONDD
         JSR CHRGET
         JMP FNAME
SDONEE
         CMP #'"
         BNE SDENEE
         JSR CHRGET
         JMP FNAME
NAMSTR
         LDY #0
         STY LENGTH
         LDA LOCATE        ; RESTORE BASIC POINTER
         STA $7A
         LDA LOCATE+1
         STA $7B
SDENEE
         JSR CHRGOT        ; GET CHAR
         JSR $AD9E         ; GET STRING
         BIT $0D           ; IS IT A STRING ?
         BMI NSTRNG        ; YES BRANCH
         JMP SYNTAX        ; NO SYNTAX ERROR
FLNRRA
         JMP FLNERR
NSTRNG
         JSR $B6A6
         STA STRLEN
```

```
        CLC
        ADC $22
        STA STRGND
        LDA #0
        TAY
        STY YSTORE
        ADC $23
        STA STRGND+1
SFNAME
        LDY YSTORE
        LDA ($22),Y
        CMP #',
        BEQ SNXTVR          ; GET NXT VARIABLE
        LDY LENGTH
        STA NAME,Y          ; SAVE FILE NAME CHAR
        LDY YSTORE
        INY
        CPY STRLEN
        BNE SFNMA           ; END OF FILENAME EXIT
        INC LENGTH
        JMP SDONE
SFNMA
        INC YSTORE
        INC LENGTH
        CPY #31             ; FILENAME TO LONG ?
        BCS FLNRRA          ; YES ERROR
        JMP SFNAME          ; DO AGAIN
SNXTVR
        LDA $7A
        STA BASLOC
        LDA $7B
        STA BASLOC+1
        LDA #1
        STA STRFLG          ; NON-ZERO STRING FLAG
        CLC
        TYA
        ADC $22             ; SET BASIC POINTER
        STA $7A             ; TO STRING POINTER
        LDA #0
        ADC $23
        STA $7B
        JMP NXTVRA
SNEXT
        LDA $7B
        CMP STRGND+1
        BCC SNEXTA
        BEQ SNEXTB
        BNE SNEXTC
SNEXTA
```

```
        JSR CHRGOT
        JMP NEXTA
SNEXTB
        LDA $7A
        CMP STRGND
        BCC SNEXTA
SNEXTC
        LDA BASLOC
        STA $7A
        LDA BASLOC+1
        STA $7B
        LDA #0
        STA STRFLG
        JSR CHRGOT
        JMP NEXTA
NMSTRA
        JMP NAMSTR
        NXTVRA
        JMP NXTVRB          ; GET NEXT VAR
BLODE                       ; LOAD BINARY PROGRAM
        CMP #148
        BNE BLODAA
        JSR CHRGET
        JMP BSAV
BLODAA
        CMP #138
        BNE BLODAB
        JSR BLODAC
        JMP (START)
BLODAB
        CMP #147
        BEQ BLODAC
        JMP SYNTAX
BLODAC
        JSR CHRGET
        LDY #0              ; RESET SAVE FLAG
        STY SVEFLG
        STY EXEFLG
        STY YSTORE
        STY LENGTH
        STY $0A
BLODEA
        SEC                 ; SET DEFAULT VALUES
        LDY $7A             ; GET BASIC POINTER
        STY LOCATE          ; AND SAVE
        LDY $7B
        STY LOCATE+1
        LDY #1
        STY SECOND          ; SECOND ADDRESS IS 1
```

```
        LDY #8
        STY DRVNUM          ; DRIVE NUMBER IS ONE
        LDY #0              ; RESET POINTER
FNAME
        CMP #'$             ; STRING VARIABLE ?
        BEQ NMSTRA          ; YES GET STRING
        CMP #',
        BEQ NXTVRA          ; GET NXT VARIABLE
        CMP #0
        BEQ DONE            ; END OF LINE EXIT
        CMP #':
        BEQ DONE            ; END OF LINE EXIT
        CMP #'"
        BNE FNAMEA
        JSR CHRGET
        JMP FNAME
FNAMEA
        LDY LENGTH
        STA NAME,Y          ; SAVE NAME CHAR
        INY
        CPY #31             ; FILENAME TO LONG ?
        BCS FLNERR          ; YES ERROR
        INC LENGTH
        JSR CHRGET          ; GET NEXT CHAR
        JMP FNAME           ; DO AGAIN
FLNERR
        LDA #6              ; SET POINTER
        JMP GETERR          ; OUTPUT ERROR MESSAGE
CHCKNM                      ; SAVE AND CHECK FILENAME
        LDY LENGTH
        CPY #0              ; LENGTH
        BEQ FLNERR          ; ERROR FILE NAME
        CPY #31
        BCS FLNERR          ; ERROR FILE NAME
        STY LENGTH          ; FILENAME LENGTH
        RTS
DONE
        JSR CHCKNM          ; SET FILENAME LENGTH
DONEA
        LDA SVEFLG          ; CHECK SAVE/LOAD
        BEQ DONEB           ; TO LOAD/EXEC/COMMAND
        JSR SAVE            ; SAVE COMMAND
        JMP CHRGOT          ; DONE RETURN
DONEB
        LDA EXEFLG
        BEQ DONEC
        JMP EXECTA
DONEC
        JSR LOAD            ; LOAD FILE
```

```
        JMP CHRGOT          ; BACK TO BASIC
HEXADS                      ; GET HEXIDECIMAL ADDRESS
        JSR STRING          ; CONVERT TO NUMBER
        LDX VALU            ; SAVE START ADDRESS
        LDY VALU+1
        STX SRTADS
        STY SRTADS+1
        JMP ADRESA          ; SET SECONDARY ADRESS
ADRESS
        JSR CHRGET          ; GET NEXT CHAR
        CMP #'$             ; CHECK HEX ADDRESS
        BEQ HEXADS          ; YES GET
        JSR $AD8A           ; GET ADDRESS
        JSR $B7F7           ; FP TO INT
        STY SRTADS          ; SAVE START ADDRESS
        STA SRTADS+1
ADRESA
        LDX #0              ; SECONDARY ADDRESS
        STX SECOND          ; TO ZERO-NEW ADDRESS
        LDA #0              ; RESET ADDRESS FLAG
        STA ADSFLG
        JSR CHRGOT          ; GET LAST CHAR
NEXT                        ; CHECK FOR NEXT VARIABLE
        LDY STRFLG          ; IN STRING ?
        CPY #0
        BEQ NEXTA           ; NO BRANCH
        JMP SNEXT           ; YES
NEXTA
        CMP #',             ; COMMA
        BEQ NXTVAR          ; GET NEXT VARIABLE
        CMP #0              ; END OF LINE
        BEQ DONEA           ; YES THEN END
        CMP #':
        BEQ DONEA
        CMP #'"
        BNE NEXTAA
        JSR CHRGET
        JMP NEXTA
NEXTAA
        JMP SYNTAX          ; NO SYNTAX ERROR
NXTVRB
        JSR CHCKNM          ; CHECK FILENAME
NXTVAR                      ; CHECK VARIABLES
        JSR CHRGET          ; GET NEXT CHAR
        CMP #'A             ; NEW ADDRESS
        BEQ ADRESS          ; YES GET VALUE
        CMP #'S             ; NEW SLOT ?
        BEQ VOLUME          ; YES THROW AWAY
        CMP #'D             ; NEW DRIVE ?
```

```
          BEQ DRIVE        ; YES GET VALUE
          CMP #'V          ; NEW VOLUME ?
          BEQ VOLUME       ; YES THROW AWAY
          CMP #'L          ; LENGTH ?
          BEQ LENTHA       ; YES GET VALUE
          CMP #'R
          BEQ VOLUME
          JMP SYNTAX       ; NO SYNTAX ERROR
DRIVE                      ; GET DRIVE NUMBER
          JSR CHRGET       ; GET NEXT CHAR
          CMP #'$          ; HEX FORMAT ?
          BEQ HEXDRV       ; YES GET VALUE
          JSR GETBYT       ; NO GET VALUE
          JSR DRIVEA       ; SET DRV NUMBER
          JMP NEXT
DRIVEA                     ; SET DRIVE
          CPX #1           ; DEFAULT IS ONE
          BEQ DRVONE       ; SET TO DRIVE ONE
          CPX #2           ; SECOND DRIVE
          BNE ERRDRV       ; OUTPUT ERROR
          LDX #9           ; SET DRIVE TWO
          STX DRVNUM
          RTS
ERRDRV
          LDX #9           ; ILLEGAL DEVICE
          JSR $A437        ; OUTPUT ERROR
          JMP CHRGOT       ; BACK TO BASIC
DRVONE                     ; SET TO DRIVE ONE
          LDX #8
          STX DRVNUM
          RTS
HEXDRV                     ; GET HEX DRIVE NUMBER
          JSR STRING       ; CONVERT TO NUMBER
          LDX VALU         ; GET DRIVE NUMBER
          JSR DRIVEA       ; SET DRIVE NUMBER
          JMP NEXT         ; GET NEXT VARIABLE
VOLUME                     ; DISREGARD VOL AN SLOT
          JSR CHRGET       ; GET NEXT CHAR
          CMP #'$          ; HEX FORMAT ?
          BEQ HEXVOL       ; GET VALUE
          JSR GETBYT       ; GET VALUE DROP IT
          JMP NEXT         ; GET NEXT VARIABLE
HEXVOL                     ; GET HEX VALUE
          JSR STRING       ; CONVERT TO NUMBER
          JMP NEXT         ; GET NEXT VARIABLE
LENTHA                     ; LENGTH OF BINARY PROGRAM
          JSR CHRGET       ; GET NEXT CHAR
          CMP #'$          ; HEX FORMAT ?
          BEQ HEXLNG       ; GET VALUE
```

```
         JSR $AD8A          ; GET LENGTH
         JSR $B7F7          ; FP TO INTEGER
LENTHB
         STY ENDADS         ; SAVE LENGTH
         STA ENDADS+1
         LDA #0             ; RESET LENGTH FLAG
         STA LENFLG
         JSR CHRGOT         ; GET LAST CHAR
         JMP NEXT           ; GET NEXT VARIABLE
HEXLNG
         JSR STRING         ; CONVERT TO NUM
         LDY VALU           ; GET LENGTH
         LDA VALU+1         ; SAVE LENGTH
         JMP LENTHB
LOAD                        ; LOADS FILE
         LDA #8             ; LOGICAL FILE NUMBER
         LDX DRVNUM         ; FLOPPY DEVICE NUMBER
         LDY SECOND         ; 1=OLD ADD 0=NEW ADD
         JSR $FFBA          ; SET FILE PARAMETERS
         LDA LENGTH         ; GET LENGTH
         LDX #<NAME         ; POINT TO NAME
         LDY #>NAME
         JSR $FFBD          ; SET FILE NAME
         LDA #0             ; LOAD FLAG
         LDX SRTADS         ; GET START ADDRESS
         LDY SRTADS+1
         JSR $FFD5          ; LOAD FILE
         BCS LOADC          ; ERROR OCCURED
         JSR $FFB7
         AND #$BF
         BEQ LOADA
         JMP DISERR
LOADA
         LDA $7B
         CMP #$02
         BNE LOADB
         JMP $A474
LOADB
         RTS                ; NO RETURN
LOADC
         JMP $E0F9          ; ERROR JUMP
STRING                      ; ASCII HEX STRING TO INT
         LDY #$FF
STRNGA
         JSR CHRGET         ; GET FIRST DIGIT
         CMP #'"
         BEQ STREND
         CMP #',
         BEQ STREND         ; COMMA EXIT
```

```
        CMP #00
        BEQ STREND       ; END OF LINE EXIT
        INY
        CPY #4
        BCS ERRSTR       ; STRING ERROR ROUTINE
        STA STRVAL,Y     ; SAVE HEX STRING
        JMP STRNGA       ; DO AGAIN
STREND
        JSR CONVRT       ; CONVERT TO NUM
        JMP CHRGOT       ; RETURN
ERRSTR
        JMP $B248        ; ILLEGAL QUAN
CONVRT                   ; CONVERTS ASCII TO HEX
        SEC
        CPY #5           ; CHECK TO LONG OF STRING
        BCS ERRSTR       ; OUTPUT ERROR
        CPY #$FF
        BEQ ERRSTR       ; OUTPUT ERROR
        STY LENSTR       ; SAVE LENGTH
CNVRTA
        LDA STRVAL,Y     ; GET LS DIGIT
        CMP #$3A         ; CHECK FOR LETTER
        BCC DIGIT        ; NO 0-9
        SBC #7           ; CORRECT LETTER
DIGIT
        SEC
        SBC #$30         ; ASCII TO NUMBER
        CMP #16          ; CHECK A-F
        BCS ERRSTR       ; OUTPUT ERROR
        STA STRVAL,Y     ; SAVE VALUE
        DEY              ; DECREMENT COUNTER
        BPL CNVRTA       ; GET NEXT DIGIT
        LDX LENSTR       ; DONE TRUCATE FROM
        LDA STRVAL,X     ; 4 BYTES TO TWO
        DEX              ; GET LSBDIGIT & DEC COUNTER
        BMI ENDA         ; CONVERSION DONE
        ASL STRVAL,X     ; CORRECT NEXT LSB
        ASL STRVAL,X     ; MULTIPLY BY 16
        ASL STRVAL,X     ; TWO BYTES TO ONE
        ASL STRVAL,X
        CLC
        ADC STRVAL,X     ; ADD LSB
        STA VALU         ; SAVE VALUE
        DEX              ; DECREASE COUNTER
        BMI ENDB         ; CONVERSION DONE
        LDA STRVAL,X     ; GET NEXT LSB
        DEX              ; DEC COUNTER
        BMI ENDC         ; CONVERSION DONE
        ASL STRVAL,X     ; CORRECT MSB
```

```
        ASL STRVAL,X        ; MULTIPLY BY 16
        ASL STRVAL,X        ; FROM TWO BYTES
        ASL STRVAL,X        ; TO ONE BYTE
        CLC                 ; ADD TO NMSB
        ADC STRVAL,X        ; AND SAVE VALUE
        JMP ENDC
ENDA
        STA VALU            ; SAVE LS BYTE
ENDB
        LDA #0              ; ZERO MS BYTE
ENDC
        STA VALU+1          ; SAVE MS BYTE
        RTS
SYNTAX
        JMP $AF08
BSAV                        ; BSAVE COMMAND
        LDY #0
        STY LENGTH
        STY YSTORE
        STY EXEFLG
        LDY #1              ; SET SAVE FLAG
        STY SVEFLG          ; SET ADDRESS FLAG
        STY ADSFLG          ; SET LENGTH FLAG
        STY LENFLG
        JMP BLODEA          ; GET PARAMETERS&SAVE
SAVE                        ; SAVE FILE ROUTINE
        LDA ADSFLG          ; CHECK ADDRESS VAR
        BNE SYNTAX          ; NO ADDRESS ERROR
        LDA LENFLG          ; CHECK LENGTH
        BNE SYNTAX          ; NO LENGTH ERROR
        CLC                 ; GET END ADDRESS
        LDA SRTADS          ; GET START ADDRESS
        ADC ENDADS          ; GET LENGTH
        STA ENDADS          ; SAVE LSB END ADRESS
        LDA SRTADS+1        ; GET START ADDRESS
        ADC ENDADS+1        ; GET LENGTH
        STA ENDADS+1        ; SAVE MSB END ADRES
        LDA #1              ; LOGICAL FILE NUMBER
        LDX DRVNUM          ; GET DRIVE
        LDY SECOND          ; GET SECONDARY ADRESS
        JSR $FFBA           ; SET FILE PARAMETERS
        LDA LENGTH          ; GET LENGTH FILENAME
        LDX #<NAME          ; POINT TO FILENAME
        LDY #>NAME
        JSR $FFBD           ; SET FILENAME
        LDA SRTADS+1        ; GET START ADDRESS
        STA $FC             ; SAVE MSB
        LDA SRTADS          ; GET START ADDRESS
        STA $FB             ; SAVE LSB
```

```
        LDA #$FB          ; POINT TO ADDRESS
        LDX ENDADS        ; GET ADDRESS
        LDY ENDADS+1
        JSR $FFD8         ; SAVE FILE
        BCS SAVEA         ; ERROR BRANCH
        LDA $90
        BNE SAVEA
        RTS
SAVEAA
        PLA
SAVEA
        JMP GTRYB         ; DISPLAY ERROR
EXECUT
        PHA
        LDA #0
        STA $90
        LDA EXEFLG        ; EXEC IN EFFECT ?
        BEQ EXCTA
        LDA #0
        STA EXEFLG
        STA EXFFLG
        STA EXGFLG
        LDA #5            ; YES CLOSE PRESENT FILE
        JSR CLOSE
        JSR CLRCH
EXCTA
        PLA
        LDY #0            ; RESET SAVE FLAG
        STY EXEFLG
        STY EXFFLG
        STY EXGFLG
        STY SVEFLG
        STY YSTORE
        STY LENGTH
        LDY #1            ; SET EXEC FLAG
        STY EXEFLG
        JMP BLODEA
EXECTA
        LDY #0
        STY $90           ; CLEAR STATUS
        LDA LENGTH        ; LENGTH OF FILENAME
        LDX #<NAME        ; ADDRESS OF FILE NAME
        LDY #>NAME
        JSR SETNAM        ; SET FILE PARAMETERS
        LDA #5            ; LOGICAL FILE NUMBER
        LDX DRVNUM        ; DEVICE FLOPPY NUMBER
        TAY               ; SECONDARY ADDRESS
        JSR SETLFS        ; SET FILE PARAMETERS
        JSR OPEN          ; OPEN FILE FOR READ
```

```
        LDX #5          ; SET EXEC FOR INPUT
        JSR CHKIN
        LDA $9D         ; DIRECT MODE ?
        BNE EXECAA      ; YES BRANCH
        RTS             ; NO BACK TO BASIC PROG
EXECAA                  ; EXECUTE EXEC FILE
        LDA #13         ; TO BEGINNING OF LINE
        STA 631
        JSR CHROUT
        SEC
        JSR CURSOR      ; GET CURSOR LOCATION
        STX PLACE       ; SAVE LOCATION
        STY PLACE+1
        JMP LOOPA
RETURN
        LDX #0          ; SET FOR INPUT
        JSR CHKIN
        JSR EXEC        ; EXEC LINE
RTURNA
        LDA #0
        STA EXGFLG
        STA EXFFLG
        LDA #13
        STA 631
        JSR CHROUT
        SEC
        JSR CURSOR      ; GET CURSOR LOCATION
        STX PLACE       ; SAVE LOCATION
        STY PLACE+1
        LDX #5          ; SET FOR INPUT
        JSR CHKIN
LOOPA
        LDA #0
        JSR BASIN       ; GET NEXT CHAR
        CMP #13         ; END OF LINE ?
        BEQ RETURN      ; YES EXECUTE LINE
        LDY $90
        CPY #0
        BNE NDFILE      ; END ROUTINE
        CMP #0          ; NO CHAR ?
        BEQ NDFILE      ; YES END ROUTINE
        JSR CHROUT
        JMP LOOPA       ; DO AGAIN
EXEC
        LDA #0
        STA EXFFLG
        STA EXGFLG
        CLC
        LDX PLACE
```

```
          LDY PLACE+1
          JSR CURSOR
          LDA #1              ; TELL KERNAL
          STA 198
          JSR $A560           ; GET LINE INTO BUFFER
          STX $7A
          STY $7B
          JSR CHRGET
          TAX
          BNE EXECB
          RTS
EXECB
          LDX #$FF
          STX $3A             ; SIGN FOR DIRECT MODE
          BCC EXECA           ; NUMBER INSERT LINE
          JSR $A579           ; CHANGE TO INTERPERTER
          LDA #1
          STA EXGFLG
          JMP ($308)          ; EXECUTE COMMAND
EXECA                         ; INSERT LINE
          TAX
          PLA                 ; REMOVE RETURN ADDRESS
          PLA
          TXA
          PHP
          PHA
          LDA #1              ; SET INSERT FLAG
          STA EXFFLG
          PLA
          PLP
          JMP $A49C           ; INSERT LINE
NDFILE
          LDA #0
          STA EXEFLG          ; RESET EXEC FLAG
          STA EXFFLG          ; RESET EXEC FLAG
          STA EXGFLG          ; RESET EXEC FLAG
          LDA #5              ; LOGICAL FILE #
          JSR CLOSE           ; CLOSE FILE
          JSR CLRCH
          JSR CHRGOT
          JMP MAIN            ; BACK TO BASIC
DTNA
          JMP DISERR
DISERR                        ; GET DRIVE ERROR
          LDA $90             ; GET STATUS
          CMP #64             ; IGNORE IF END OF FILE
          BNE GTRYB
          RTS
GTRYB
```

```
        JSR CLRCH          ; RESET DEFAULT DEVICES
        LDA #13            ; OUTPUT LF TO SCREEN
        JSR CHROUT
        LDA #8             ; DEV # OF DISK
        STA $BA
        JSR $FFB4          ; SEND TALK
        LDA #$6F           ; SEND SEC ADDRESS
        STA $B9
        JSR $FF96          ; SEND SEC FOR TALK
GTRYA
        JSR $FFA5          ; READ BYTE
        JSR CHROUT         ; DISPLAY TO SCREEN
        CMP #13
        BNE GTRYA          ; DO UNTIL DONE
        JSR $FFAB          ; SEND UNLISTEN
RTS                        ; DONE
                           ; FILL ZERO'S TO $CC00
                           ; FOR CHECKSUM ACCURACY
        .BYTE 0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .BYTE 0,0,0,0,0,0
        .END
```

VITA

Lonald L. Fink was born on October 9, 1954 in New Albany, Indiana. He attended Floyd Central High School and entered Purdue University in West Lafayette, Indiana in August, 1972. He graduated with a Bachelor of Science degree from Purdue University in May 1976. He entered Speed Scientific School, University of Louisville, Kentucky, in August, 1984. The author is expected to receive his Master of Engineering degree with Specialization in Electrical Engineering in May, 1988.